1992-06

Cost estimation of software development and the implications for the program manager

Doyle, Glenn Cameron

Monterey, California. Naval Postgraduate School

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) 36	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS Program Element No / Project No / Task No / Work Unit Accession Number

11. TITLE (*Include Security Classification*)
COST ESTIMATION OF SOFTWARE DEVELOPMENT AND THE IMPLICATIONS FOR THE PROGRAM MANAGER

12. PERSONAL AUTHOR(S) Doyle, Glenn C.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*year, month, day*)	15. PAGE COUNT
Master's Thesis	From To	June 1992	148

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (*continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUBGROUP	

19. ABSTRACT (*continue on reverse if necessary and identify by block number*)

Cost estimation of computer software development is a critical problem for the Department of Defense. The aquisition of major weapons or hardware has been impacted by cost overuns and schedule slippage in software development. Program Managers are responsible for estimating a program's cost using the information provided by the contractor and the cost analysis divisions within the System Commands. This study first analyzes why variance exists between the different estimates for the same software project that are provided to the Program Manager as input to the budget estimate. The study then examines four methods that are used to understand and reduce the variance between the estimates to give the Program Manager more control over the software development cost estimation process. A set of five specific decision rules is developed for the Program Manager to implement in the cost estimation process. The intent of the study is to improve the accuracy of the cost estimate by reducing the variance between the independently generated estimates submitted to the Program Manager.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Joseph San Miguel	22b. TELEPHONE (*Include Area code*) (408) 646-2187 22c. OFFICE SYMBOL AS/Sm

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

Cost Estimation of Software Development and the Implications for the Program Manager

by
Glenn Cameron Doyle
Lieutenant, United States Navy
B.S., The Pennsylvania State University, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
June, 1992

ABSTRACT

Cost estimation of computer software development is a critical problem for the Department of Defense. The acquisition of major weapons or hardware has been impacted cost overruns and schedule slippage in software development. Program Managers are responsible for estimating a program's using the information provided by the contractor and the cost analysis divisions within the System Commands. This study first analyzes why variance exists between the different estimates for the same software project that are provided to the Program Manager as input to the budget estimate. The study then examines four methods that are used to understand and reduce the variance between the estimates to give the Program Manager more control over the software development cost estimation process. A set of five specific decision rules is developed for the Program Manager to implement in the cost estimation process. The intent of the study is to improve the accuracy of the cost estimate by reducing variance between the independently generated estimate submitted to the Program Manager.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisors, Dr. Joseph San Miguel and Dr. Michael Sovereign, for their expert guidance and cooperative assistance. I would also like to recognize and thank Dr. Thomas Frazier of the Institute for Defense Analysis for suggesting the mathematical combination of independent cost estimates.

I would also like to acknowledge and thank CDR Larry Cable, NAVAIRSYSCOM, for identifying this project as an area of potential research and for providing funding for the research travel.

A special thanks goes to my wife, Joy, for typing, proofing, and especially for taking care of everything else while I was working on the thesis.

I. INTRODUCTION

A. OBJECTIVES

Cost estimation of computer software development is a critical problem for the Department of Defense (DOD). The DOD spends billions of dollars annually on computer resources. Over the last 35 years, the costs of software development have surpassed hardware development costs and now dominate the cost of computer resources (Figure 1) [Ref 1:p. 18]. Errors or inaccuracies in estimating software development costs will result in large differences between expected and actual costs in the final product.

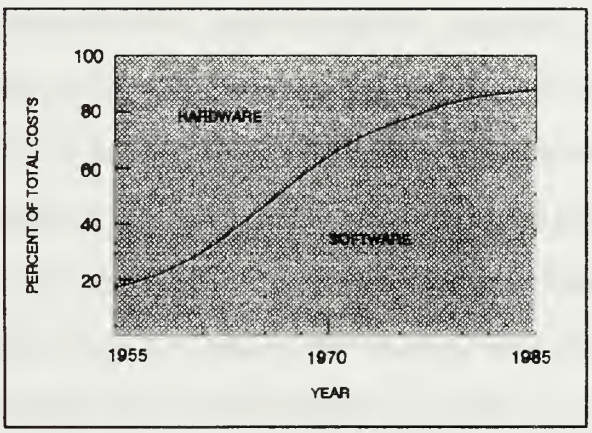

Figure 1. Software versus Hardware Development Costs

Software development has posed significant problems for management as computers play an increasingly important role in

modern weapon systems as technology advances. In the early 1950's, all weapon systems were designed with analog technology. In the mid 1950's and 1960's, digital subsystems were introduced and rapidly incorporated into weapon systems. In the 1970's, there was exponential growth in the field of electronics with advances in integrated circuits, introduction of microprocessors, and the need to counter the Soviet threat with quality and electronically sophisticated weapons. During the 1980's, the engineering trend was the total digitalization of weapon systems.

Defense systems have grown increasingly sophisticated primarily as a result of advancing computer technology. Early weapon systems employed software mainly for monitoring the condition of the hardware and used less than 10,000 lines of code. Modern weapon systems rely on software programs for command, control, and communication interface with hardware systems. Current projects use software for the operation and control of equipment, including signal processing and fire control systems. The size of these software projects often exceed one million lines of code. [Ref. 2:p. 11] Software, therefore, must be highly reliable and fault tolerant.

Technology has evolved to the point where most of the development effort is in software while the hardware has become increasingly standardized. With continued advancement in the development of electronic equipment in the 1990's, management will find software costs an increasingly

significant part of the total lifecycle costs of weapons systems. DOD faces the challenge of producing high quality and sophisticated weapon systems that are greatly dependent upon rapidly evolving software technology.

The dynamic nature of software development has presented tremendous difficulties for cost analysts. However, accurate and timely information regarding software development costs is critical for the budget process as cost estimation plays a vital role in program management and the Planning, Programming, and Budgeting System (PPBS). The program manager (PM) is responsible for creating a budget and executing a program within that budget. He/she must be able to estimate software cost with a reasonable degree of confidence to responsibly plan and manage a program.

Software development within the Federal Government and private enterprise has been impacted by cost and schedule overruns. In addition, delivered end products have not met the customers needs in terms of capability and quality. It has been difficult to accurately project software costs due to problems in assessing accurate input data, a rapidly evolving and complex software environment, lack of adequate and available historical information, and the labor intensive nature of software development. Software project cost overruns are not uncommon and are frequently accompanied by significant schedule slippage.

A recent example of a Navy program affected by a software development cost overrun is the S-3B program. According to the Cost Performance Report (CPR) released by the contractor in September of 1987, software categorized as "Avionics Software" was 8.26 percent over budget. This category alone resulted in a $2.8 million cost overrun. It is difficult to identify all of the software development costs in the S-3B program because software costs are aggregated into higher level hardware costs in certain items. However, it appears that the total software development cost for the program (approximately $80.7 million) exceeded the budgeted cost (approximately $75.2 million) by over $5 five million.

The difficulty with estimating software development costs is deriving quantitative relationships between the measurable front end requirements and the output costs and schedules. These estimates of costs and schedules must be timely and accurate. Many factors can cause a final cost to be significantly different from an initial cost estimate that was reasonable when it was originally developed. Among these factors are operational requirements that were changed, incomplete, or submitted late. Also, contractors that manage their sub-contractors poorly have difficulty with cost control. A contractor's business base that decreases will result in increased overhead and expenses and ultimately higher costs for the purchaser. Parts, materials, and government information delivered late will slow down the

schedule and increase costs. The reduction of DOD program budgets may require the project to be reconfigured and the final cost adjusted accordingly.

There are also many reasons that the initial development cost estimate itself may be inaccurate. The key element in most software cost estimates is the size estimate of the project. The size, or numbers of lines of code (LOC) in many instances, must be predicted early in the acquisition cycle when requirements are neither firm nor fully defined. Another factor that affects the initial cost estimate is the choice of computer language used. Higher order languages (HOL) such as Ada and Fortran will have different costs per line than assembly languages such as INTEL 8086 Assembly and ATAC-16M Assembly. Variations in methods and standards for counting lines of code differ between estimators and will affect the cost estimate. Code condition, whether the code is new, modified, or re-used, is also an important element in the costing of a project. Additionally, the development environment (mainframe versus microprocessor), type of software (system, application, or support), and accuracy of the operational requirements are also components in an accurate estimate. The organizations developing the estimates must have clear assumptions concerning the methods used and the inputs considered when developing a cost estimate. In Chapter IV, the factors affecting the software development cost estimate are further explored.

This research study examines the difficulties in software cost estimation and the implications for the PM. Estimating techniques from private contractors, Naval Air Systems Command (NAVAIRSYSCOM), and the Naval Center for Cost Analysis (NCA) were analyzed to determine:

1. Why does variance exists between cost estimates generated by independent cost estimators for the same project?
2. How to reduce variance between cost estimates generated by independent estimators?

Variance is defined in this study as the difference in cost between the estimates generated by independent organizations for the same project.

Sensitivity analyses is conducted to identify elements in the estimate that have a particularly large effect on creating variance between the cost estimates. Identifying these elements is key to bringing the estimation process into control.

Specific recommendations are then made to reduce variance between the independent estimators and thereby reduce the risk and the uncertainty for the manager. Additionally, a set of decision rules is presented for the PM to assist him/her in making a reasonable budget estimate and to provide a means to validate a contractor's estimate.

The two main benefits of the study are:

1. At the Program Manager level - Effective budgeting is critical for a manager of a major program. Accurate and

timely cost estimates will greatly benefit the manager in his projections of real time and future requirements for funding. A clear understanding of the difficulties in software cost estimation and a set of tailored decision rules will assist the manager in making critical budget decisions for a software development project.

2. Within the DOD - It is plausible to assume that the defense budget will continue to decrease in the upcoming years. Scrutiny in all aspects of program development will be imperative throughout the DOD. An improved process for cost estimation, particularly for an area as elusive as software development, will provide benefits to the financial management of programs throughout DOD.

This research study will improve the process of cost estimation of software development by determining why variance exists between independent estimates of the same project. The study will also recommend how these variances may be reduced to assist the Program Manager in reconciling these independent estimates into a composite estimate that best represents the most likely cost of the software development project.

B. SCOPE

The principles presented in this thesis are applicable to software development managers inside, as well as outside, the federal government. The DOD is one of the largest single users of computer technology and this study primarily reflects DOD policies and techniques. The research questions are directed towards the manager entrusted with budget responsibility and is designed to provide decision making guidance for preparing cost estimates for software development.

The nature of current software cost estimation is highly technical in terms of mathematics. This study maintains a top-level emphasis and provides technical breakdowns to the level necessary to understand parametric relationships from a management perspective. Due to limitations in research time and access to operating programs, computer models for generating software development costs are not comprehensively discussed. A brief description of available models is provided in Chapter III along with a comparison of their capabilities and limitations.

Specific software development costs have been difficult to extract from existing DOD reports including the Contractor Cost Data Report (CCDR), Cost Performance Report (CPR), and Selected Acquisition Report (SAR). Software development costs are typically embedded into higher levels of the work breakdown structure (WBS) and cannot be extracted from total software costs. Currently, however, as PMs are more aware of software costing problems, new programs have specifically requested the contractor to delineate these costs. In the future it should be easier to obtain and evaluate software development costs.

C. METHODOLOGY

The research was conducted in three overlapping stages. First, data were collected through books, papers, prior theses' at the Naval Postgraduate School (NPS), and a key word

search of "cost estimation" and "software". Interviews were conducted with local experts from the Naval Postgraduate School within the Administrative Science and Operation Research Departments. Phone interviews were conducted with personnel from a variety of organizations within the DOD and are appropriately documented in the reference section.

Second, a research trip was conducted to Washington D.C. to interview key organizations and to collect empirical data. Primary organizations of interest were NAVAIRSYSCOM, NCA, Institute for Defense Analysis (IDA) and the Navy Comptroller Office. A comprehensive and personal overview on the various organization's methodologies and philosophies was acquired as well as an assessment of their current capabilities and future requirements.

Third, all collected data in steps one and two were compiled and analyzed. Actual and potential problem areas within the software costing arena were examined. Variance between cost estimating techniques from different organizations were identified and presented. Based on these variances, decision rules were formulated for the PM. These decision rules are designed to aid the PM in developing a reasonable budget estimate. The decision rules will also help the PM to validate the contractors estimate and thereby reduce the uncertainty and the risk of costing software development.

D. RESEARCH OVERVIEW

In Chapter II, the framework for understanding the software development process is established. Management of software development projects is discussed from a historical perspective and from the official DOD position as set forth by current instructions. Methods for cost estimation are presented with the advantages and disadvantages discussed for each technique. The process for estimating the size of the software project is addressed since size is typically the critical input to the cost estimate.

In Chapter III, software cost estimation models are discussed. A general approach to modeling is presented as it applies specifically to software development. The study analyzes five of the most popular and frequently used automated models in terms of their capabilities, strengths, weaknesses, and availability. The five models discussed include the Software Lifecycle Model (SLIM), the Jensen System-4 model, the Software Productivity, Quality, and Reliability Estimator (SPQR\20), the Constructive Cost Model (COCOMO), and the Software Architecture and Sizing and Estimating Tool (SASET) model.

In Chapter IV, the first research question is addressed, "Why variance exists between cost estimates generated by independent cost estimators for the same project?" The question is addressed from two perspectives: variance caused by methods used to develop the estimates and variance caused

by inconsistent assumptions between estimators on the project's initial parameters.

In Chapter V, the second research question is addressed "How to reduce variance between cost estimates generated by independent estimators?" The analysis discusses techniques for reconciling estimates generated by independent sources for the same project, procedures for reducing differences in initial assumptions on the project's parameters, and DOD organizations and tools. The result of this analysis is that the program manager will be better equipped to improve the process of cost estimation and reduce the variance between independent software development cost estimates.

In Chapter VI, the conclusions of the study are presented in a concise format and the results of the data analysis are restated. Recommendations for the Program Manager are stated in the form of five decision rules that can be applied to a improve the process of cost estimation of software development by answering <u>why</u> variance exists between independent estimates and <u>how</u> to reduce the variance between the independent estimates.

II. BACKGROUND OF SOFTWARE DEVELOPMENT COST ESTIMATION

Cost estimation of software development has been a difficult problem for the DOD since the beginning of the computer age. Established procedures have existed for hardware cost estimation and analysts have attempted to apply the hardware estimation techniques directly to software development projects. Although estimators had limited success with estimating the costs of hardware equipment, the same techniques had poor results with software project cost estimates.

This chapter examines how cost estimation for software development has evolved from the early days of computers to the current techniques and instructions comprising the software development field today.

A. SOFTWARE DEVELOPMENT PROCESS

The software development process is divided into functional phases that delineate distinct areas in a project development cycle. A process divided into phases provides the manager information to exercise control over the resource requirements throughout the software systems lifetime. The software development process is discussed in this section from a historical perspective of lifecycle resource requirements to

the current perspective as set forth by the military standard establishing the phases of software development projects.

1. Lifecycle Resource Requirements

Lifecycle phases are functional phases that include all resource requirements from concept exploration to operations and support. The user (customer) of software needs to know the projected resource requirements for manpower, cost, schedule and critical milestones in order to prepare an economic analysis to determine funding. Phases in the software development process help to categorize those resource requirements to improve management control and cost estimation.

Early efforts in lifecycle management were put forth by Peter Norden of IBM in the 1970's. From a large data base of evidence, Norden empirically deduced that manpower requirements for research and development (R&D) projects follow the lifecycle pattern formulated by Lord Rayleigh. This pattern is composed of overlapping work cycles, or phases, and are well described by the Rayleigh equation for manpower (Equation 1).

$$Y' = 2Kate^{-at^2} \quad (1)$$

where:

Y' = Manpower utilized each time period
K = Total cumulative manpower utilized by end of project
a = Shape parameter governing time to peak manpower
t = elapsed time from start of cycle

The magnitude and duration of the phases have stable and predictive structures that can be exploited for project planning and control. The cycles are:

1. Proponent Planning and Functional Specification
2. Design and Coding
3. Test and Validation
4. Extension
5. Modification
6. Maintenance

Norden then linked the cycles to create a project profile. Figure 2 shows the individual cycles laid out in their respective time relationships. When the individual cycles are added together they produce the profile of the entire project called the "project curve." The project curve has useful applications for projecting the out-year budget, manpower, and schedule milestone requirements. Note that the project curve has a long, one sided tail that explains the

phenomenon of why project work can be 90 percent complete in work but only 67 percent complete in time.

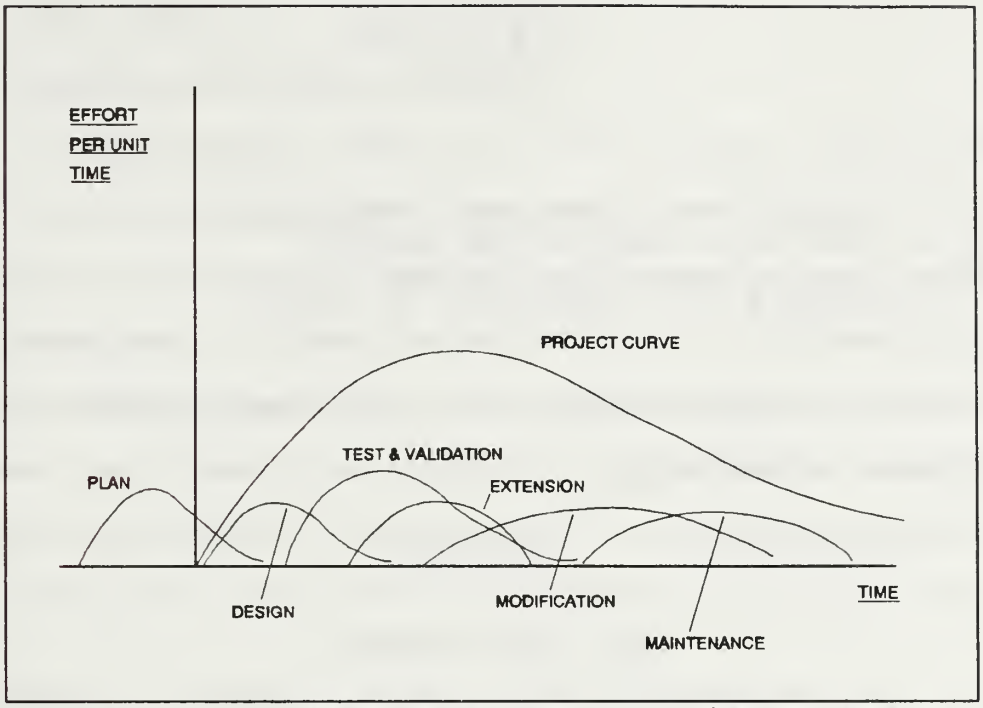

Figure 2. Software Development Project Phases as presented by Peter Norden

In the late 1970's, Lawrence Putnam, President of Quantitative Software Management, Inc., specifically applied the work of Norden to software projects. Putnam related software system attributes (number of files, reports, application sub-programs, and source statements) to development resource requirements including manpower, total effort, cost, and project duration. [Ref. 3:p. 16] The Rayleigh equation was rewritten by Putnam with the shape

parameter (a) expressed in terms of time to reach peak effort in a software development process (Equation 2).

$$Y' = \frac{K}{Td^2} t e^{-t^2/2Td^2} \qquad (2)$$

where:

Y′ = manpower utilized each time period
K = total work done on the system
Td = time to reach peak effort
t = elapsed time from start of cycle

The shape parameter, (a), has been replaced by a new expression (Equation 3).

$$a = K/Td^2 \qquad (3)$$

The expression acts as a "software development difficulty" (SDD) variable in terms of programming effort. Putnam examined and plotted 40 systems and determined that when SDD was small it corresponded to easy systems, when SDD was large it corresponded to difficult systems, and there appeared to be a continuum in between [Ref. 3:p. 20]. For the software development industry, the difficulty remained to accurately estimate the value of the key parameters in the equation: lifecycle effort (K) and development time (Td).

Putnam also redefined the phases of the lifecycle model to specifically address software development. The four phases of the Putnam Model are:

1. Systems definition
2. Functional Design and Specification
3. Development
4. Operations and Maintenance

Additionally, there is a fifth phase, Installation, that overlaps the end of the Development phase and the beginning of the Operations and Maintenance phase (Figure 3). The Putnam model is based on and is similar to Norden's model. However, the Putnam model was specifically designed for software development and has useful applications in applying the Rayleigh manpower relationship to software projects.

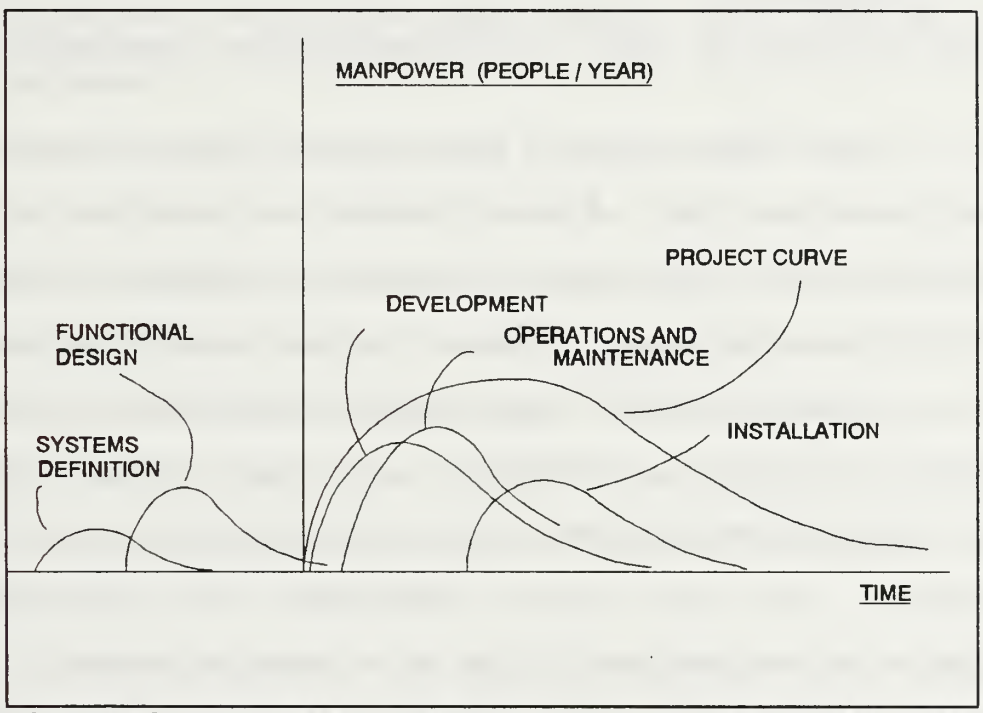

Figure 3. Software Development Project Phases as presented by Lawrence Putnam

Putnam presented several key management implications based on these concepts. With a constant manpower effort, work is wasted early in the project, effort is inadequate during peak requirements, and schedule slippage results based on the original linear prediction of manpower effort required. It was customary that management first estimated the number of source lines of code (SLOC) and then apply a historical productivity rate (PR) to determine the total man-year effort required. For example:

SLOC = 600,000 lines of code (estimate)
PR = 1,000 SLOC/MY (man year) (historical)

Therefore:

Development Effort (DE) = 600,000 SLOC/1,000 SLOC/MY = 600MY

DE was then linearly distributed across the predicted project duration (P1). Figure 4 shows the resulting manpower distribution and the areas of excess or deficient manpower resources resulting from a linear application of manpower.

Putnam performed least squares, best fit correlation analysis to try to determine if there was a linear relationship between productivity and delivered source lines of codes. To this point, management had assumed that productivity and delivered SLOC were closely related. His database consisted of over 400 projects collected at the United States Air Force's Rome Air Development Center with a

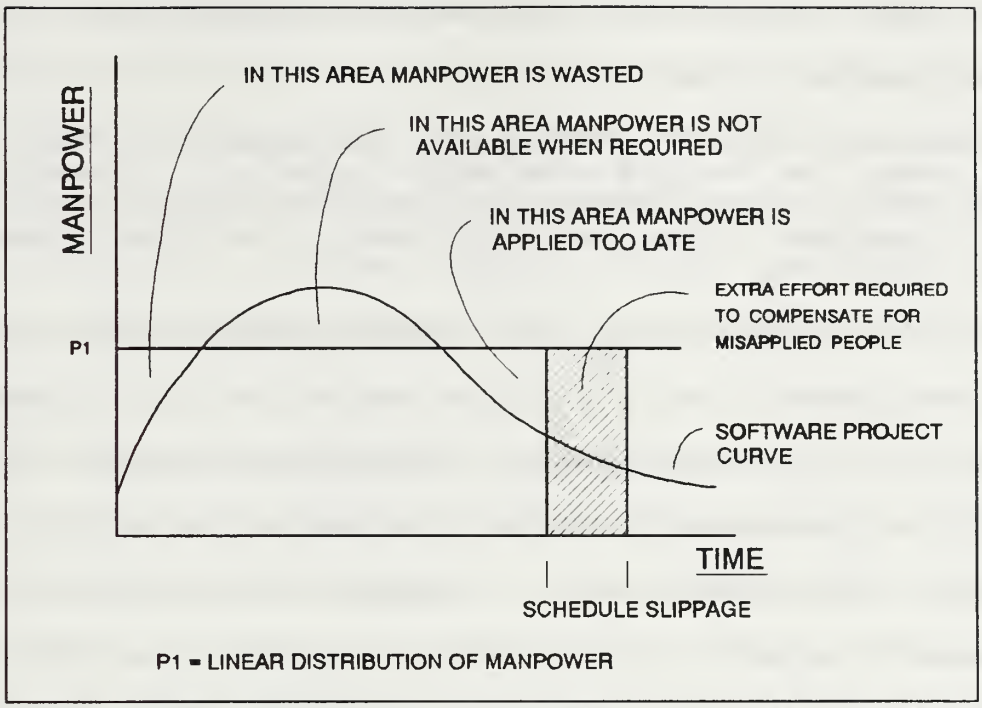

Figure 4. The Non-linear Nature of a Software Development Project versus a Linear Application of Manpower

wide range in system size (100 to 1,000,000 SLOC), project duration (1 month to 6 years), man-months of effort (1 to 20,000 MM), average number of people (1 to 500 people), and productivity rate (10 to several thousand SLOC/MM). The resulting correlation coefficient (R) was found to be R = 0.033415 which demonstrates virtually no correlation between productivity and delivered source lines of code [Ref. 4:pp. 29-30].

The conclusion is that management must dedicate manpower resources in accordance with the particular needs of

the project phases and not as an averaged distribution. Putnam further states:

> The money may have been spent, the people may have been on board, but because some people were not in phase with the demand of the system, their effort was wasted and must be reapplied later at increased cost and greater time.[Ref. 3:p. 71]

Putnam also concluded that management cannot shorten the development time of a system without severely increasing the difficulty of the project. Development time is the most sensitive parameter. There is a natural software development schedule that results in a minimum cost solution. If management chooses to compress the schedule, they can expect significantly higher costs and possibly schedule overruns exceeding the original, natural schedule [Ref. 3:p. 75].

Man-months are typically used as a means of measuring the size of the job, but man and months are not interchangeable. The time, or numbers of months required, is a factor of sequential constraints. The maximum numbers of men that can be used at any one time depends upon the number of independent sub-tasks available. Fred Brooks, author of "The Mythical Man-Month," stated, "Adding manpower to a late software project makes it later." [Ref. 5:p. 25] This conclusion is widely accepted and known as "Brooks Law."

A development process that is divided into phases can be controlled by management because each phase has specific requirements that must be met before the following phase can be started. The DOD has incorporated the phase development ideas of Rayleigh, Norden, and Putnam into a Military Standard that delineates the phases of the software development process.

2. Department of Defense Standard 2167A

DOD has a Military Standard publication, "Defense System Software Development", that divides the software development process into eight phases (Figure 5). This standard, DOD-STD-2167A, was designed to establish requirements that provide insight for the PM, as well as other government officials, into a contractor's software development and testing and evaluation efforts.

The requirements of the standard must be applied during the acquisition, development, and support process of software systems. It provides the authority for the PM to establish, evaluate, and maintain quality in software and associated documentation by requiring the contractor to work within prescribed guidelines. These guidelines are comprehensive and include management, engineering, formal qualification testing, product evaluation, configuration management, and the transition to software support.

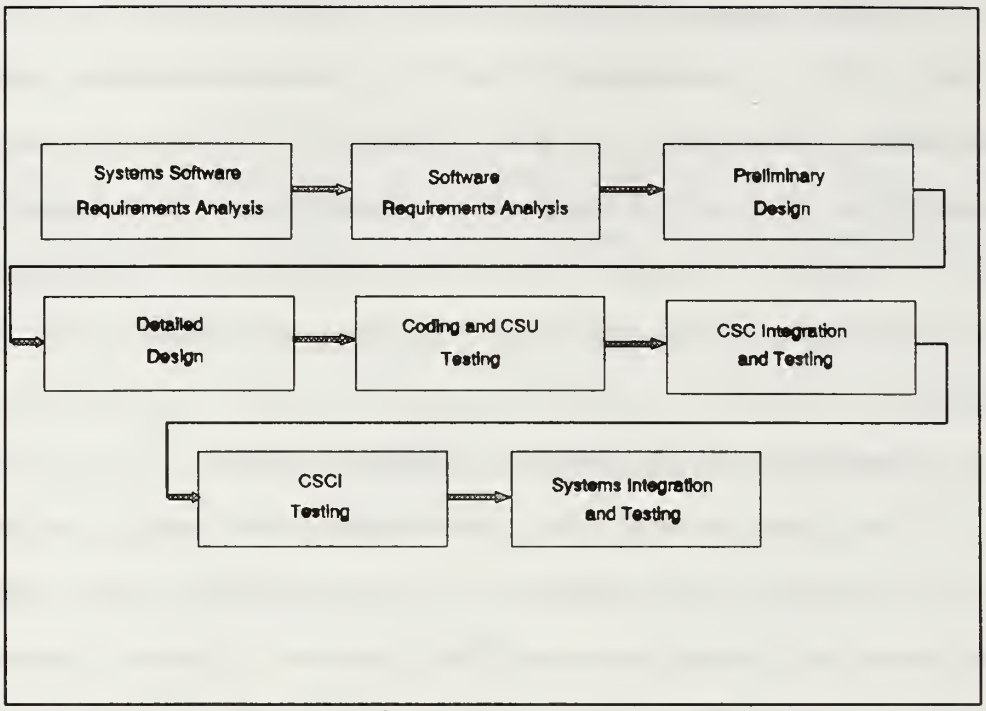

Figure 5. DOD-STD-2167A PHASES

The process begins at the aggregate level and is divided into work element levels. Elements are then coded and rebuilt into the total end product. An abbreviated work breakdown structure is provided (Figure 6) to display the total software development system as a product orientated, family tree relating the elements of work to each other and to the end product.

In the first phase, Systems Software Requirements Analysis, the requirements for the Computer Software Configuration Items (CSCI) are defined and analyzed at the system level. A CSCI is at a high level in the WBS, representing an aggregate of many smaller elements. The

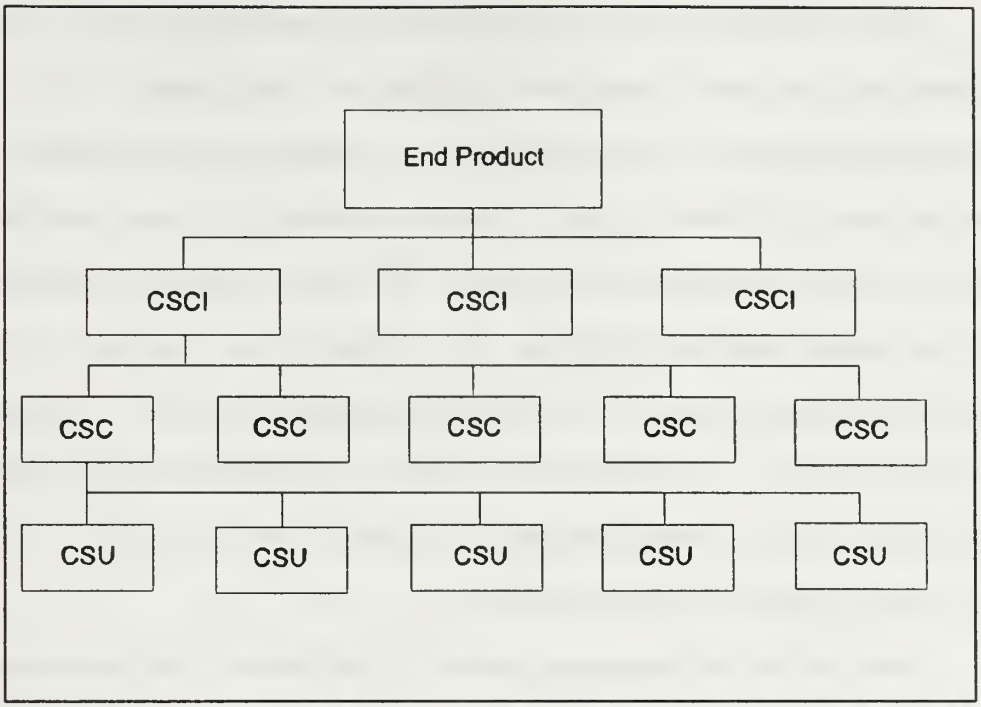

Figure 6. Software Development Work Breakdown Structure

contractor is required to analyze the CSCI's preliminary system specifications for conformance with the software requirements. The contractor shall also define a preliminary set of engineering and interface requirements for each CSCI.

The next phase, Software Requirements Analysis, continues in the preliminary analysis process by performing one or more Software Specification Reviews (SSR) in accordance with MIL-STD-1521. The contractor is now required to define a complete set of engineering and interface requirement for each CSCI.

The Preliminary Design phase assigns specification requirements to each Computer Software Component (CSC) which is a sub-component of the CSCI. Design requirements are established for each CSC. The contractor is required to conduct one or more Preliminary Design Reviews (PDR's) in accordance with MIL-STD-1521. The contractor is also required to establish test requirements for conducting CSC integration and testing and to identify formal qualification tests to comply with the requirements identified in the Software Requirement Specification (SRS).

The Detailed Design phase distributes CSC requirements to their sub-components, Computer Software Units (CSU's). The contractor shall establish test requirements, responsibilities, and schedules for testing all CSU's. The Detailed Design phase is intended to ensure readiness for operational coding.

The Coding and CSU Testing phase creates the source and object code for CSU's. The contractor ensures that code algorithms and logic employed by each CSU is technically correct and satisfies all specified requirements. The contractor is also required to test all CSU's, ensuring that the test procedures are developed properly and documented comprehensively. Revisions shall be made to design documentation and code based on the results of the CSU tests.

In the CSC Integration and Testing phase the coded CSU's are reintegrated into CSC's. The contractor ensures that the CSC's algorithms and logic are correct and that each CSC satisfies its specific requirements. CSC's are tested and the necessary revisions to design documentation and code are made to bring the CSC's to specification levels. The contractor shall also conduct one or more Test Readiness Reviews (TRR) as required by MIL-STD-1521.

In the CSCI Testing phase the CSC's are integrated into CSCI's. The CSCI's are tested for conformance to all operational and specification requirements. Preparations are made for the Functional Configuration Audit (FCA) and the Physical Configuration Audit (PCA). All CSCI test results are documented in the Interface Design Document (IDD) and revisions are made to the CSCI's to bring them up to functional and physical specifications.

In the final phase, Systems Integration and Testing, the CSCI's are combined into the total system which is the end product. Extensive testing is conducted including the Functional and Physical Configuration Audits. All support documentation is tested and revised for delivery. The final product is complete as an aggregate of all the sub-components and has been formally tested for acceptance.

The phases of the software development process provide a valuable management control tool that promotes enhanced front end management visibility. The PM will have formal access to development manpower cost, schedule, and critical milestones of the highly complex software development process.

B. SOFTWARE COST ESTIMATION TECHNIQUES

Central to any valid estimating process is an organized procedure. The following is a seven step approach for software cost estimation taught by the Naval Center for Cost Analysis (NCA) in their introductory Software Costs Analysis course.

1. Establish objectives
2. Plan for data and requirements
    a. Purpose
    b. Products and schedules
    c. Procedures
    d. Responsibilities
    e. Required resources
    f. Assumptions
3. Define software requirements
4. Work out as much detail as possible
5. Use different estimating techniques
6. Compare and iterate estimates
    a. Exam rationale for different results
    b. Consider best, worse, and most likely case
    c. Review estimates

7. Follow-up cost estimate
    a. Track inputs and outputs
    b. Update software as project progresses

With a comprehensive process in place, estimating techniques can be selected. The methods used to estimate software development costs originate from and are similar to the methods used to estimate hardware costs (i.e. aircraft, missiles, ships). Techniques can be categorized into five general methods; parametric, analogy, top-down, bottom-up, expert judgment and price to win [Ref. 6:p. 33].

1. **Parametric Estimation**

Parametric estimation can be defined as the use of one or more cost estimating relationships (CER's) that generate development effort as output. Development effort may be in terms of either cost or man-months and is an output value of one or more input variables considered to be cost drivers. The strength of parametric methods is that they are objective, repeatable formulas. It is efficient for creating output and for sensitivity analysis. Most parametric CER's can be calibrated to the specific requirements of the development project by tailoring the database that determines the mathematical relationships.

A significant weakness of a parametric relationship is that it is always based on past data and experience. Since

forward extrapolation is required for current projects, the past data may not reflect new technologies or difficulties. Managers must remain aware that the development effort estimated by these models are based on subjective inputs. The output from an objective model should not be considered a final answer in itself but one input into the estimation process. Current examples of common parametric estimating tools are automated models such as Constructive Cost Model (COCOMO), Price-S, and Software Lifecycle Model (SLIM).

2. Analogy Estimation

Analogy is another cost estimating method. The resources required for a new project are estimated by comparing the new project to similar past projects. Inflationary corrections can be made to bring the costs up to current dollars. It is likely, however, that the historical database does not contain the same level of technology as the new program, particularly in a field that evolves as quickly as software development. This is a limitation that will exist for any estimation process that is based on historical data. An analogy should be considered an important part of the estimation process when there is cost data available on similar, past programs.

3. Top-Down Estimation

The top-down approach to software cost estimating starts with a rough order of magnitude cost estimate generated from the "big picture" requirements of the software project. The estimated resources are then divided among the components of the project. This technique provides a focus from the system level which is often the level that the manager in control of the funding is operating from. This technique requires less resources up front to produce the estimate than other methods. The tradeoff to this economy is that the estimate is not detailed at the smaller elements of the work breakdown structure. Often the level of accuracy will be based on the subjective judgment and experience of the manager and may be subject to large, unpredictable errors. SLIM and COCOMO are good examples of a top-down estimating approach.

4. Bottom-Up Estimation

The opposite approach to top-down estimation is bottom-up. Bottom-up methods estimate each component in the WBS separately at the engineering level and then sums the component estimates to produce an estimate of the total project. The advantage of a bottom-up technique is that the estimate is very detailed and may be quite accurate. It also provides individual commitment and accountability to the costs of the components which facilitates a follow-up estimate as

actual cost data becomes available. Clearly the bottom-up estimate requires significant, up front resources to produce such an extensive cost breakdown.

The type of project that is being estimated will be a factor in deciding whether to use a bottom-up estimate or not. Projects that use a high percentage of existing technology (with known costs) can be estimated with a reasonable outlay of resources. New technology, however, where each component has a high level of uncertainty, may generate a bottom-up estimate where the uncertainties have combined multiplicatively. The result will be an estimate that may be less accurate and more expensive than other lower cost techniques.

5. Expert Judgment

Expert judgment is the process of relying on the experience of specialists to obtain a software development cost estimate. Expert judgment represents the knowledge, in the optimal case, of the best minds in the field. This provides a particular advantage if the project is a brand new concept where there is little or no historical data to use as a reference point. The experts will rely on their judgment based on past experience to make the best estimate of cost. This technique is, however, no better than the level of expertise of the participants. Even the most objective and

comprehensive professional will be subject to personal biases and incomplete recall. The caliber of the "expert" is dependent on program need, fiscal resources, and the availability of the expert. Expert judgment always has a place in the cost estimation process as a subjective, expert opinion is a valuable asset to a PM.

6. Price-to-Win

A Price-to-Win estimate is developed by determining the price believed necessary to win the contract and designing the estimate to that price. This is in contrast to the analytical sizing approach where cost is the output of several input variables such as size, previous productivity, and complexity factors. In price-to-win, the estimating process is driven first by the desired expected cost and the input variables are manipulated to produce that desired output. [Ref. 7:pp. 2-3]

A contractor that needs work to stay in business must make a competitive bid. If the contract risk is slanted toward the government instead of the contractor, as in a cost plus fixed fee (CPFF) contract, getting the award is the prime concern of the contractor, knowing that he will be able to adjust the final price later if required.

Clearly there are problems with this technique. The estimator's focus is on outputs (cost) vice the inputs. This

results in an estimate that was created lacking rigor and discipline. The risk of error under this condition is very high and will be difficult to predict. Price-to-win estimating confuses software cost estimation with bidding for contract award. However, this technique is a reality and PM's must take steps to control it. Contracts should be written so that the risk is not fully absorbed by the government. Additionally, the PM should use multiple cost estimates to prevent a single Price-to-win estimate from eroding the baseline budget of a program.

C. SOFTWARE SIZING

The output of the estimation process is only as sound as the critical inputs are accurate. The majority of cost estimating methods rely on sizing as the initial and most significant input parameter. This is a difficult parameter to estimate because the estimate is formulated early in the acquisition cycle when the operational requirements (OR's) may not be firmly or fully defined. Additionally, the project's system architecture is still volatile to changes based on funding availability, a changing threat assessment, and the engineering capability to produce the hardware as specified. Software sizing is discussed in three parts: language

considerations, line of code definition, and sizing techniques.

1. Language Considerations

The type of computer language used in a project must be known to accurately predict size. Assembly languages such as INTEL 8086 Assembly, ATAC-16M Assembly, PDP-11 Assembly, and VAX 11/780 Assembly are written in machine code. Higher Order Languages (HOL) such as Ada, Fortran, and BASIC are structured to read like English. A HOL command is three times the size (in line of code) of a similar command in assembly language and uses considerably more memory. In terms of total programmer effort, however, a HOL command is also about three times simpler to write than an assembly language command. The net effect is that the total programmer man-months between HOL and assembly languages are comparable [Ref. 8]. A sizing input, however, would erroneously cost the larger HOL programs three times higher than assembly language programs unless there is a sizing correction factor in place. Assumptions on language type must be understood by the estimators before a meaningful comparison of independent estimates can be made.

2. Line of Code Definition

The counting rules for size, line of code (LOC) for example, is also a factor in determining the sizing input of

a software cost estimate. There are many variations in how lines of code are counted. The more common methods are:

1. Count only executable lines.
2. Count executable and data definition lines.
3. Count executable, data definition, and comment lines.
4. Count executable, data definition, comment, and job control lines.
5. Count lines as physical lines on an input screen.
6. Count lines as terminated by logical delimiters.
7. Count lines as a combination of the above methods.

The total quantity of LOC that is used in the sizing parameter will be a function of the counting method. The PM, contractor, and independent estimator must understand which counting method will be used so that they are developing their estimates under the same assumptions.

3. Sizing Techniques

These are several different techniques in practice to estimate the size of the software project. Size is the key parameter in accurately forecasting software development cost and needs to be carefully addressed by the estimator. The categories of sizing techniques are parallel to the techniques used in estimating total software development cost and include

analogy, regression, function points, expert judgment and parametric methods.

a. Analogy

Analogy techniques estimate the LOC size of the current software project by comparing it to similar, past programs. This is useful early in a program when general capability specifications exist but the operational requirements are not defined sufficiently to allow a detailed estimate.

b. Structural Decomposition

Structural decomposition is a bottoms-up approach that is reliable when there is significant data available for the development project. The structure is decomposed into the smallest elements practical and individual LOC estimates are produced for each element [Ref. 6:p. 37]. The summation of these predictions will give a total size estimate for the project. Although this technique is resource intensive, it produces a reliable size estimate and is recommended when there is valid data available.

c. Parametric

Parametric sizing methodology is useful early in the lifecycle of the project. Regression analysis can provide a reasonable LOC size estimate assuming that the historical data base has:

1. high correlation
2. low standard error
3. existing data points including current technology

Since all data bases are historical there will always be some extrapolation, however, this is acceptable as long as the above three conditions are reasonably met.

d. Expert Judgment

Expert judgment models are estimates produced, based on experience, by specialists in the software development field. It is advantageous to use more than one expert as resources allow. These predictions can be useful early in a program, particularly if there is not valid historical data available to implement other techniques. The disadvantage of the expert judgment technique is that with the estimate of size comes certain biases the expert may have.

e. Function Points

Function point models depart from the common practice of estimating software size by counting lines of code. This study will look at the International Business Machine (IBM) function point model as a simple example [Ref. 6:p. 13]. In this function point model, size is estimated from five functional program attributes:inputs, outputs, inquiries, master files, and interfaces.

The number of each of the attributes is summed and then calibrated for complexity to give a Function Point Total (FPT). The FPT is then adjusted by a Language Expansion Constant (LANG) which is an expression of size requirements tailored to individual languages. For example, Ada has a LANG of 72 and Fortran has a LANG of 105. The product of FPT and LANG is the size of the software project. Examples of function models in current use are the Software Productivity, Quality, and Reliability Estimator (SPQR/20) and the Software Architecture Sizing and Estimating Tool (SASET) model, both of which consider significantly more attributes and have more sophisticated size equations than the IBM example.

An advantage of function point models is that they are available for size prediction earlier in the lifecycle than LOC models. The function points represent the user's vision of the software project and the associated attributes. The model may be more accurate than LOC methods because of the specific attention to project design attributes in the early stages.

The major disadvantage of the function point models is that the choice of weights for the attributes and calculation of technical complexity factor were subjectively determined. It is also difficult to define the basic counts

of the attributes in constant, quantified units that can be objectively input into the model.

Even with the risk associated with these limitations, function point models are establishing a firm place in the software estimation process. The function point model has particular merit in the estimation process since it may be the only sizing method that is not based on counting lines of code and can be done accurately early in the project lifecycle.

Sizing a software development project is critical to getting a valid estimate of cost. The industry trend, using any combination of the discussed sizing techniques, is to underestimate the size and cause a cost understatement. This tendency is caused by unfounded optimism by the contractor and program manger, incomplete recall of past experiences, and/or unfamiliarity with the requirements of the total project.

D. SUMMARY

This chapter has presented a broad background of the software development process. A manager that divides the development process into functional phases has increased control over the project as it evolves. The early pioneers in software development management, including Rayleigh, Norden,

Putnam, and Brooks, understood this and developed techniques and formulas to optimize project development.

The military has standardized software development by issuing DOD-STD-2167A. This standard established phases and test requirements that provide increased oversight for management into a contractor's software development and test and evaluation efforts.

The process of estimating software was then discussed. A seven step approach developed by the NCA provides a organized procedure to streamline cost estimation. With a comprehensive procedure in place, estimating techniques can be selected. Six common software cost estimation techniques were defined with a brief description of the strengths and weaknesses of each technique.

Estimating the size of the software project was then addressed as the key input to most software cost (effort) and schedule models. Language considerations, line of code definition, and five sizing techniques were studied as they applied to estimating size in the software development process.

In the following chapter, software cost estimation models will be discussed. A general approach to modeling is presented as it applies to software development. The study analyzes five of the most popular and frequently used

automated models in terms of their capabilities, strengths, weaknesses, and availability.

III. COST ESTIMATION MODELS

A. GENERAL

Cost estimation models are tools that a manager of a software development project can use to make reasonable predictions of the resources required to develop a project. The need for an effective and efficient model for estimating software cost has been recognized by managers from the early days of software development.

The first estimation tools were simple "paper and pencil" models. Development effort and schedule requirements were calculated based on a simple set of equations derived from small, historical databases. Estimates of project size initially came from counting boxes filled with punch cards and have evolved to the more sophisticated techniques discussed in the previous chapter. As the databases became more comprehensive and the estimators gained experience, equations were reformulated and expanded to consider additional factors such as complexity of software, type of project, and programmer's experience.

As models became increasingly sophisticated, the mathematics required to support them became more complex and cumbersome. Concurrently, however, computers became increasingly common in the office and laboratory as digital

technology advanced. With the increased accessibility of computers and computational power, a variety of automated cost models were developed to assist the software cost estimator. Currently, there are over 25 automated cost models in use by various government agencies throughout the DOD, National Aeronautics and Space Administration (NASA), and the military industrial community [Ref. 9:p. 3-3].

Cost models have multiple applications for the manager of a project. Early in the program, models provide estimates of expected cost, schedule, and manpower requirements. Models may also provide a means for the manager to independently validate estimates provided by contractors or elements within his/her own organization. Models have additional application during later development phases as the program evolves. Operational requirements or available resources may change and the model may be used for weighing tradeoff options within the new constraints of the project.

Models typically operate from the same fundamental arrangement. The general model structure (Figure 7) consists of five sequentially executed stages: Sizing, Manpower, Schedule, Manloading, and Cost [Ref. 10:p. 4].

Virtually every software cost model starts with an estimate of size. Size may be determined based on parametrics, analogy, function points or expert judgment as discussed in Chapter II. Most cost models, however, require

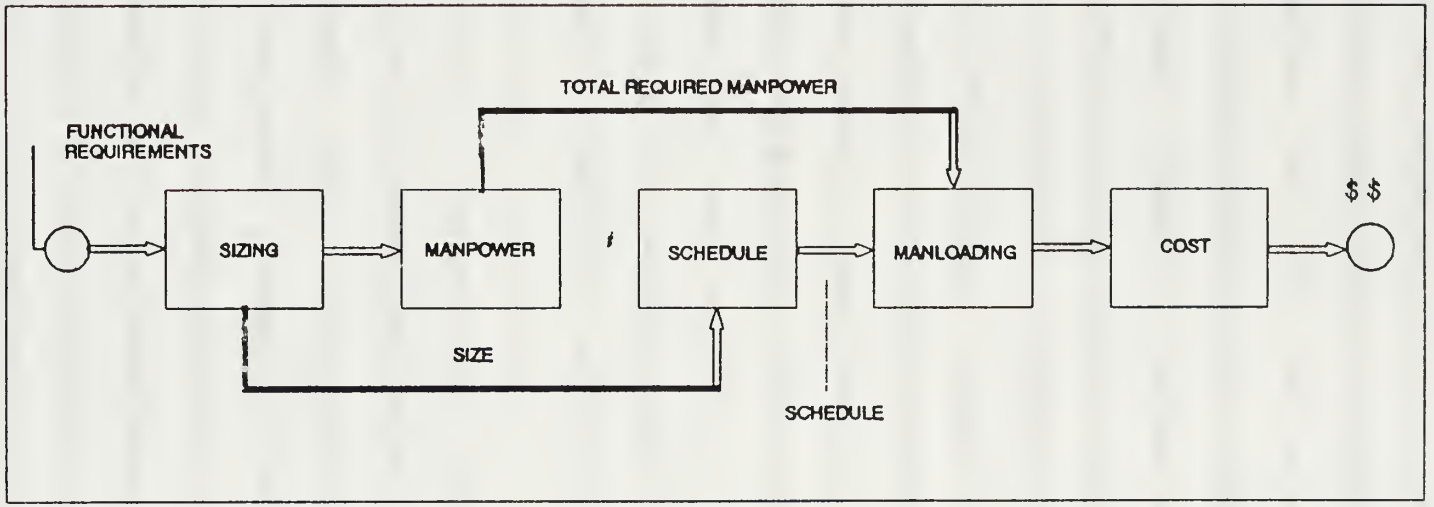

Figure 7. The General Model Structure

an LOC input for size rather than calculate a size estimate. This puts the program manager in the position of roughly accepting the contractor's size estimate since size is difficult to estimate in the early stages of the acquisition cycle. The Naval Center for Cost Analysis has started to do some independent sizing as their database improves, but they are limited in resources and often use the contractor's size estimate for their Independent Cost Estimate (ICE)[Ref. 8].

Manpower required for the project is typically calculated from the sizing input. A cost estimating relationship (CER) correlating manpower effort to software size is derived based on a historical data base. There are usually additional parametric factors that are considered and act as multipliers to tailor the projected manpower requirement to the data base. Examples of these factors include project complexity, personnel experience, software language, and required reliability.

The schedule phase serves to spread the size estimate across the software development phases to produce a project schedule. The manloading phase then spreads the manpower prediction from the manpower phase over the optimal schedule provided by the schedule phase. Recall from Chapter II that the manpower requirement for the project is not linear but rather follows a Rayleigh curve distribution.

The cost phase generates dollar figures for the effort expended. Not all models include this final phase but rather

leave it to the manager to consider the complexities of the total cost. There are many factors that must be considered in pricing an entire job including direct development costs, indirect and overhead expenses, and the present value of money for lengthy projects.

There are many variations of the general model that have strengths in different areas. Various models may not emphasize the same development phases or activities of interest. It is important to select a model that is particularly suited to the current project, has fidelity in the value of the estimate, is well documented so the user has confidence in the model's methodology, and is user friendly for the cost analyst who only spends a portion of his/her time operating cost models.[Ref. 1:p. 476]

In addition to these criteria, the PM must be concerned about cost and access. Estimation models can be purchased, leased or time-shared and range from no cost (Government provided) to a lease price of $25,000 per year. The model of choice should be able to run on standard DOD equipment, which is currently the Zenith 248. The Zenith 248 is an IBM compatible personal computer (PC) that is common in most DOD offices and laboratories. It incorporates a 80286 processor and is by most standards an outdated hardware system. There are some DOD provided PC's with 80386 processors available but they are not yet common in the government work place. Other arrangements exist for the PM, such as telephone hook-ups or

time-sharing a large mainframe, but these options create additional complications for the PM that may not be offset by the benefits of the particular model. Cost and access information will be further explored in this chapter as particular software cost models are addressed.

In summary, there is no "best" model as all models have strengths and weaknesses. There is, however, a common risk in over relying on any model. Although inputs to models are subjective, managers may mistakenly assume outputs to be objective and express overconfidence in the results [Ref. 8]. Models do, however, provide a powerful tool to the cost estimator and should be an integral part of the cost estimation process.

B. AUTOMATED MODELS

1. Software Lifecycle Model

The Software Lifecycle Model (SLIM) is owned and maintained by Quantitative Software Management (QSM), Inc. The program was developed by Lawrence Putnam in 1978 based on his previous work in software lifecycle management. The program has been continually updated and is widely accepted by industry as an effective software cost estimating tool.

The model is based on the Rayleigh manpower distribution function. There are three primary inputs to the model: lines of source code, the technology constant, and development time. Source lines of code must be directly input

by the user. The technology constant can be input directly by the estimator or calculated by SLIM based on the database. Development time must be input and is critical to the SLIM model because of the heavy emphasis the Rayleigh function places on schedules and schedule deviations.

SLIM will provide estimates of required personnel, costs, projected cash flow, limiting constraints, and risk. It is also useful for sensitivity analysis. Effort and schedule tradeoffs can be conducted but manpower effort is significantly influenced by small changes in schedule as expected from the Rayleigh manpower equations. The model is "white box," indicating that the equations and database are published. The SLIM equations for system size and expenditure rate are as follows:

System size

$$Ss=(Ck)K^{1/3}Td^{4/3} \quad (4)$$

where:
- Ss = Number of delivered source instructions
- K = Lifecycle effort in man-years
- Ck = Technology constant
- Td = Development time in years

Expenditure rate

$$Y'=K/Td^{2}te^{-t^{2}/2Td^{2}} \quad (5)$$

where:
- Y' = Expenditure rate
- K = Total lifecycle effort
- Td = Development time in years
- t = Time in years

The Slim model has many strengths. One is that it is IBM compatible and will run on the standard DOD PC. The program is user friendly and easy to apply. SLIM's output is easy to read and it's graphics capability is outstanding. The SLIM model places specific emphasis on the relationship between development effort and total time required.

A weakness of SLIM is also related to the development effort and time relationship. The Rayleigh equation forces a severe tradeoff of development effort to schedule time that may impose too steep an effort "penalty" for small changes in schedule.

The cost of using SLIM may also be considered a disadvantage. In 1992, SLIM is available on a lease basis at a first unit cost of $25,000 per year. Additional units cost the organization an additional $1,000 per year. QSM no longer offers a special rate to DOD ($8,000 in 1986) and the high price may deter PM's from leasing the program. [Ref. 11]

2. Price-S

The Price-S model was originated by Frank Freiman and further developed by Robert Park. It is one in the family of Price models owned by RCA, Inc. Price Systems Division. The Price family of models include:

1. Price-S - estimates software development costs
2. Price-SL - estimates software lifecycle costs

3. Price-H - estimates hardware costs

4. Price-SZ - estimates software size

Price-S is a proprietorship model and is "black box" indicating that equation and database information are not available to the public. The model provides assessments of manpower, schedule, and budget requirements for software development and is designed for use over a wide variety of applications. It was originally designed for the military but has gained prominence within the government and private industry [Ref. 9:p. III-14]. The model provides output in terms of phase and elements cost, schedule, and sensitivity analysis.

There are 64 inputs for Price-S that are divided into three groups of parameters: product related, organization related and environment related. Of these 64 inputs, most have default values but eight must have direct input. Those eight are instructions, application, resources, utility, platform, complexity, new design and new code. [Ref. 12:p. 8] The model develops cost by relating input variables to development effort using CER's derived from the historical database. These results are adjusted in the calibration mode to fit the model to the specific environment of the user.

There are several strengths that make Price-S appealing. It is easy to use and to calibrate. It produces a comprehensive output particularly tailored to software

development. The Price-S model is easy to use for "what if" sensitivity analysis. The model is designed with over 20 years of experience at RCA and it is updated frequently.

A drawback of the Price-S system for a program office is it's cost. The model is available as a time-share that requires a dedicated phone line and costs $82.50 per connect hour. It is also available as a lease and costs $15,000 per year. For the leased version, a Prime 750 computer is required to run the program. There are DOD models available to the PM at no cost and therefore consideration must be given to the cost of the both these arrangements for Price-S before a decision is made on the model.

RCA has announced that there will be a IBM compatible version of Price-S available in July of 1992. There is no price data available yet but the required processing capability to run the model will exceed the current standard for DOD PC's. RCA recommends a computer with at least a 80386 processor and a math co-processor to make full use of the model's capabilities [Ref. 13].

Another weakness of the Price-S model is it's black box nature. The algorithms used are not public domain and the user cannot analyze the equations or the database to substantiate the models credibility.

3. Jensen

The Jensen model was developed in 1979 by Dr. Randall W. Jensen, Chief Scientist of Hughes Aircraft Corporation. The model is based on the Rayleigh distribution curve for effort and schedule estimates and tradeoff analysis. The tradeoff relationship between cost and schedule, however, is not as steep in Jensen models as it is in the SLIM model. The model has undergone considerable refinements over the last decade and has been revised from the original JS-1 model in 1979 to the current System-4 update.

There are four input parameter groups in Jensen models for cost estimation: size, complexity of the project, technical ability of the developer, and staff availability. These groups consist of a total of 50 input factors that can be entered either as a specific values or as ranges. The central equation to the model is:

$$Se=(Ce)TdK^{1/2} \qquad (6)$$

where:

Se = Effective software size in source lines of code
Ce = Effective developer constant
Td = Development time
K = Total lifecycle effort in man-years

The Jensen model is user friendly and easy to calibrate to the project environment. It is supported by good documentation and is "white box." Additionally, System-4 is compatible to the standard DOD PC which makes it an attractive option for a program office.

The model is leased or time-shared and proprietary control is held by Computer Economics, Inc. (CEI). The cost of System-4 may be prohibitive for some DOD users. System-4 costs the user $8,000 per year for three units and $700 per year for each additional unit to lease the program. The cost to time-share the program is $49.25 per hour. [Ref. 14]

CEI released a sizing tool in June of 1990 to supplement the System-4. This new sizing tool, called CEIS, is an analogy model that is based on mathematical advances made by Dr. Joseph Lambert, Head of Computer Science Department at Pennsylvania State University. CEIS has a relatively small database of 40 projects but is tailored toward government and military projects and the implementation of Ada.

4. Software Productivity, Quality, and Reliability Estimator

Software Productivity, Quality, and Reliability Estimator (SPQR/20) was developed by T. Capers Jones of Software Productivity Research, Inc., in 1985. The model is intended to estimate the outcome of software development projects very early in the planning stages. SPQR/20 provides output on cost, schedule, staff requirements, size, and predicted product quality. Predicted project quality includes anticipated defect levels, number of test cases and test runs required, and the effectiveness of pre-test and test activities.

The model is unique because it is one of the few models that has the capability to calculate size. Size may either be calculated as an output of the Albrecht Function Point technique or it may be directly input by the user. The Albrecht Function Point technique, similar to the IBM function point model discussed in Chapter II, is a comprehensive sizing tool that allows input to the model in 30 different computer languages. The accuracy of the size estimate has empirically resulted in estimates usually within 15 percent of actual size. Function point sizing is an advantage to the PM as it allows an estimate early in the acquisition cycle and it can be used to validate a LOC generated estimate.

There are 28 user inputs to the model that primarily relate to development environment, product complexity, and sizing inputs. The algorithms are black box but the proprietor has published that the database has been created from over 3000 software projects spanning over 200 organizations.

The SPQR/20 can be operated on the standard DOD PC, but the cost may be prohibitive for a program office. The first unit is available for $3,250 while additional units have a sliding cost scale based on volume [Ref. 15].

A weakness of the SPQR/20 model is that function point models tend to be highly subjective in assigning values to the function point parameters. Subjective treatment of the sizing and complexity inputs may create wide variances in output

between different estimators on the same project with the same models.

5. Constructive Cost Model

The Constructive Cost Model (COCOMO) was developed by Dr. Barry Boehm who was then Director of Software Research and Technology at TRW, Inc. COCOMO is presented and fully explained in Boehm's 1981 book, Software Engineering Economics. COCOMO is considered by many cost estimators to be the industry standard for software cost estimation models. Numerous versions of COCOMO are available and it is the most widely used software cost estimation model.

COCOMO was originally designed from a database of 63 projects representative of the major sectors of the software world including business, scientific, systems, real time and support software programs. Correlation was developed between lines of code and development effort based on the projects in the database. The estimating equations in COCOMO are not optimal least-squares fits to the data points but rather represent a line subjectively fit to the data points that best represents the relationship between size and effort [Ref. 1:p .85] . Boehm's opinion is that this line will produce less variation from the actual values than a regression line based on best fit. This relationship is the backbone of all the COCOMO models.

COCOMO is composed of a hierarchy of three increasingly detailed levels: Basic, Intermediate and Detailed. The Basic level is the simplest model. It is designed to provide a quick rough order of magnitude estimate of effort and schedule early in the lifecycle. Effort is expressed in man-months (MM) and is a function of thousands of delivered source instructions (KDSI); schedule is expressed in development time (TDEV) and is a function of MM's. Basic COCOMO is most useful for projects that are developed in a familiar environment and are small to medium size (2 - 32 KDSI). Accuracy is limited because the model does not account for specific program development factors concerning the project, personnel, or hardware. All equations used in Boehm's COCOMO models are presented in Table 1.

Intermediate COCOMO is more sophisticated and accurate than the Basic level and is the most frequently used version of Boehm's COCOMO model. Intermediate COCOMO is suitable for cost estimation in the more advanced phases of software product definition. It incorporates 15 predictor variables that represent cost s in software development. These predictor variables are effort multipliers (EM's) that calibrate the estimate to specific project characteristics and are grouped into four categories: software product attributes, computer attributes, personnel attributes, and project attributes. The EM's are multiplied against the base equation

TABLE 1. EFFORT AND SCHEDULE EQUATIONS FOR COCOMO

LEVEL/MODE	NOMINAL EFFORT	SCHEDULE
BASIC		
Organic	MM=2.4(KDSI)^1.05	TDEV=2.5(MM)^0.38
Semidetached	MM=3.0(KDSI)^1.12	TDEV=2.5(MM)^0.35
Embedded	MM=3.6(KDSI)^1.20	TDEV=2.5(MM)^0.32
INTERMEDIATE		
Organic	MM=3.2(KDSI)^1.05	TDEV=2.5(MM)^0.38
Semidetached	MM=3.0(KDSI)^1.12	TDEV=2.5(MM)^0.35
Embedded	MM=2.8(KDSI)^1.20	TDEV=2.5(MM)^0.32
DETAILED		
Organic	MM=3.2(KDSI)^1.05	TDEV=2.5(MM)^0.38
Semidetached	MM=3.0(KDSI)^1.12	TDEV=2.5(MM)^0.35
Embedded	MM=2.8(KDSI)^1.20	TDEV=2.5(MM)^0.32

(nominal) for effort and have values ranging from 0.70 to 1.65 (1.00 being the nominal default value).

Detailed COCOMO is similar to Intermediate COCOMO but has two significant improvements. First, effort multipliers are applied to each individual lifecycle phase vice across the entire project as in the Intermediate version. The phase-sensitive effort multipliers may provide a more accurate total estimate for those projects which have significant variability across phases.

Detailed COCOMO also simplifies data input procedures for large projects with many components. It provides a three-level product ranking system that groups project components into three categories: module level, subsystem level, and

system level. Effort multipliers can be applied to each of the three broad categories vice assigning cost values to every component.

Within each COCOMO model there are three development modes: organic, semi-detached and embedded. The organic mode assumes a stable development environment with very little concurrent development of hardware. There is previous experience with the project type and high productivity is assumed. There is little demand for innovative data processing architectures or algorithms and the projects are normally low cost and relatively small (less than 50 KDSI).

The semi-detached mode is an intermediate stage between organic and embedded modes. It is designed for a programmer team consisting of experienced and inexperienced personnel. Project size may be as large as 300 KDSI.

The embedded mode is used when the software must operate within tight project constraints and involves complex operating requirements with little flexibility. Examples of embedded software projects are manned space systems, avionics, and command and control systems. The embedded mode may be used for any size software project.

There are many advantages to a COCOMO model. It is user friendly, can be run on a standard DOD PC, and is relatively accurate [Ref. 16:p. 14]. COCOMO is a white box system with extensive documentation and instruction in

Software Engineering Economics. It is simple to use for sensitivity analysis and to calibrate to specific projects.

Table 2 presents the capability of the model to be within 20 percent of actual values for each level of COCOMO. Table 2 was based the relative error between the actual values for the projects in the database versus the estimated values for the projects in the database as calculated with the COCOMO model.

TABLE 2. COCOMO FIDELITY

Ability of COCOMO to predict within 20% of actual values.	
Basic	25%
Intermediate	68%
Detailed	70%

A disadvantage of COCOMO is that it does not generate a size estimate. The estimated project size must be an input to the model. An additional disadvantage to the original COCOMO model is that the database is relatively small (63 projects) and outdated (the most current project is from 1979). Fortunately, there are many COCOMO based models on the market that use the COCOMO equations and techniques and have updated databases with current project data. A few examples of COCOMO based models are:

1. WICIMO - A Pascal version developed by WANG Institute designed primarily for commercial use. The cost is a $200 one time charge for first and additional units.

2. PCOC - Designed by Eclectic Systems and implements both Intermediate and Detailed COCOMO. The cost is $850 for the first unit and $350-$175 for additional units.

3. Costmodl - Developed by the Software Technology Branch of the Johnson Space Center. It is comprised of four COCOMO versions: Basic, Intermediate, Ada-COCOMO (designed specifically for projects including Ada programming), and Keep It Simple Stupid (KISS). KISS is a user friendly derivative of COCOMO that is tailored to Mission Critical Computer Resources (MCCR's). Costmodl can be acquired for DOD users in limited quantities through the NCA at no cost.

5. Revic COCOMO - A version of COCOMO developed by the USAF and tailored specifically for MCCR projects. Revic is available at no cost for DOD users.

6. **Software Architecture, Sizing, and Estimating Tool**

The Software Architecture, Sizing, and Estimating Tool (SASET) was developed by Dr. Aaron N. Silver of the Martin Marietta Denver Astronautics Group. It was designed for the NCA under the auspices of the Office of Naval Research to estimate effort, schedule, cost, maintenance, and risk of a software development project. Additionally, SASET will also either generate a size estimate using function point techniques or allow the user to directly input project size. Comprehensive documentation for the model is available in the contract Final Report from Martin Marietta and is referenced in the bibliography.

SASET is organized into three tiers for software development cost estimation. Tier 1 addresses high-level system architecture which includes the class of software, software programming language, development schedule, and other

development and environment issues. Outputs from this tier are values representing preliminary budget and schedule multipliers. [Ref. 9:p. V-5]

Tier 2 is the sizing segment. It provides the user capability to generate project size by entering functionality aspects of the software project as input. Functions are divided into four high level categories: Processing Software, Input Tasks, Output Tasks and Security of Multi-User Systems. These categories are subdivided in specific function areas and size values are calculated for each function area based on the selected historical database. The size estimates of each function area are then compiled to produce an estimate of total project size.

Software size may also be directly input by the estimator. To input size, software must be classified as either HOL or Assembly language and then further categorized as any combination of new, modified or rehost software.

Software type must be specified as either System, Application, or Support software. System software is a collection of programs written to service other programs and is highly constrained and written in assembly language. Application software is written to process input data, solve problems, and derive outputs. It is less complex than systems data and is usually written in HOL such as FORTRAN, COBOL, and Ada. Support software is the least complex of the three and

it may include business systems, database systems, or simulation/model systems.

Tier 3 addresses software complexity issues of the hardware and software systems. There are 34 inputs that are ranked from simple to very complex and include factors such as software requirements, timing and criticality, software experience, and schedule constraints. Each of these complexity factors are subjectively determined by the estimator and input to the model to further refine the cost estimate.

SASET is user friendly, white box, compatible with a standard DOD PC, and has a comprehensive and current database. SASET has good graphics capabilities and produces output in the forms of tables, bar chart, pie chart or line graph. A notable strength of SASET is its ability to predict size and also perform all other roles normally associated with a software cost estimating tool.

A weakness of SASET is related to its strength of being a function point sizing model. Function point models have large subjectivity associated with the values and multipliers of the initial functions. However, SASET allows the user to calibrate the database to the specific project requirements which may increase the accuracy of the estimate if the user has accurate and comprehensive data.

SASET is available to DOD users through the Naval Center for Cost Analysis. There is no cost for the model and many program offices already have the program installed.

C. SUMMARY

This chapter has discussed models for estimating costs of software development. PM's can expect to have success using models in two areas. The first area is to generate software development cost estimates. Cost estimates will be used by the manager for budget input or for validating the credibility of a contractor's estimate. The second area in which models are useful for the PM is performing tradeoff studies. During project development, tradeoff studies will help to minimize the effect changes in the project requirements or resources may have on cost and schedule by enabling the manager to make an optimal decision on the tradeoff of resources.

Although there are over 25 automated models available to the PM, this study has concentrated on the five most popular and frequently used models: SLIM, Price-S, Jensen, SPQR/20, COCOMO, and SASET. There is no "best" model as each model has different strengths and weaknesses. Model selection should be based on the specific requirements of the development project and the resources available to operate the model.

The following chapter will explain why variance exists between estimates developed by independent estimators working on the same project. The COCOMO equations will be analyzed

and sensitivity analysis performed with SASET to determine which factors in a software development project have the greatest effect on cost variance between estimates.

IV. WHY VARIANCE EXISTS BETWEEN INDEPENDENTLY DEVELOPED COST ESTIMATES

Managers with responsibility for generating estimates of software development costs have implemented computers and automated cost models over the last decade to improve the accuracy of the estimate. Computers have reduced some of the uncertainty from cost estimation by introducing sophisticated equations and expansive databases into the estimation process. However, for any software project, there are still significant variance between cost estimates developed by independent cost estimation organizations. Causes of variance, those elements of a project that have the most significant effect, were identified through a series of interviews with PM's, software cost analysts, and engineers. This chapter will examine those causes to determine why discrepancies exist between independent estimates.

There is some concern among cost analysts that the elimination of variance between independent estimates is not desirable [Ref. 8][Ref. 17]. They fear that the standardization or uniformity required to eliminate these discrepancies would stifle innovation and prevent cost estimators from having the latitude necessary to comprehensively develop a valid estimate. This study is not intended to advocate restricting the managers autonomy but rather to identify cost estimation areas that are producing

variance so that a manager may make informed decisions regarding a cost estimate. The manager must be aware of what factors are causing differences between estimates when he/she is faced with reconciling multiple cost estimates to produce a consolidated cost estimate for budget submission.

For research purposes, a hypothetical software development project was created for evaluating software estimation methods and performing sensitivity analysis. Sensitivity analysis is the process of changing values of individual elements of a project so the effect of that single element can be observed on the total project outcome.

The hypothetical project for sensitivity analysis will be referred to as the "BASE" project. The BASE project was created to be similar to the SH-60B LAMPS MK III Block II Upgrade currently under development at IBM. The SH-60 Program Office at NAVAIRSYSCOM is the Department of the Navy (DON) point of contact for the program and the financial sponsor for the thesis travel conducted in support of this study. All parameters given for the Base project closely resemble the anticipated actual values of the project parameters that will be released by IBM in June of 1992 and are presented in Tables 3 and 4.

The BASE project, however, is not intended to comprehensively represent the program estimate as certain information is not available due to the sensitive nature of

TABLE 3. PARAMETERS OF THE "BASE" PROJECT

Initial Size Estimate	Class of Software	Primary Software Language	Software Type
118,500 LOC	Avionics	Ada (first-time)	Systems
HOL vs Assembly Language	Code Condition	Database selection for Calibration	
100% HOL	100% New	SASET Default (TIERS.CAL)	

TABLE 4. SOFTWARE DEVELOPMENT COMPLEXITIES OF THE "BASE" PROJECT

System Requirements	Software Requirements	Software Documentation	Travel Requirements
Fairly complete	Fairly complete	Average amount	Little travel
Man Interaction	Timing and Criticality	Software Testability	Hardware Constraints
Highly Interactive	High Reliability	Complex	Complex
Hardware Experience	Software Experience	Software Interfaces	Development Facilities
Average	Average	Some	Shared
Development vs Host Systems	Technology Impacts	Off-the-Shelf Software	Development Team
Similar	Minor	Few impacts	Experienced
Embedded Development	Development Tools	Personnel Resources	Programming Language
Little impact	Good availability	Adequate staffing	Ada

specific price information of an ongoing project. Cost values in real dollars were not included in the BASE project data and therefore the total project estimates will be expressed in terms of effort in man-months required to develop the project. The project was created for estimation with the SASET model

and all parameter choices are consistent with the input options available to the user with SASET.

Causes of variance between independent estimates can be broadly categorized into two major areas:

1. Methods used to develop the estimates
2. Inconsistent assumptions between independent estimators on project input parameters

Both of these areas will be analyzed in the remainder of this chapter with the intent of determining why these factors cause variance and to what magnitude the factors affect the total estimate of effort required to develop the software project.

A. METHODS USED TO DEVELOP THE ESTIMATE

The choice of methods for developing the software project cost estimate will directly impact the outcome of the total project estimate. Different estimation methods will emphasize different aspects of the software development process and may result in model bias and/or incomplete lifecycle coverage. The effects of model bias and incomplete lifecycle coverage must be understood by the PM to recognize why variance exists between independent estimates. This section will look at the reasons that model bias and incomplete lifecycle coverage exists as a result of the methods chosen to estimate the software development project.

1. Automated Cost Model Biases

Model bias is an emphasis by the automated cost model given to certain favored elements of the estimation process. This emphasis skews the outcome of the estimate towards the elements that are favored and may include: effort verses schedule tradeoffs, estimated size (LOC) as a primary independent variable, the weighing of certain project complexity factors, or the intended application of the software project. Each cost model inherently has model bias dependent upon the model's intrinsic equations and database. Because each model has different equations and databases, each model also will have different model bias. It is important that the PM understand the model biases for each estimate submitted to the program office in order to make a legitimate comparison among independent estimates.

An example of two models that would produce incongruous model biases would be estimates produced by the SLIM and the COCOMO model. SLIM, consistent with the Rayleigh equation, places a heavy emphasis on the tradeoff between development effort and the total time required to develop the project (Equation 7).

$$Ss=(Ck)K^{1/3}Td^{4/3} \quad (7)$$

where:
Ss = Number of delivered source instructions
K = Lifecycle effort in man-years
Ck = Technology constant
Td = Development time in years

COCOMO, on the other hand, does not address development time as a variable in the equation. The equation for effort in Intermediate COCOMO (Embedded mode) is (Equation 8):

$$MM = 2.8(KDSI)^{1.20}(\Pi EM) \quad (8)$$

where:

- MM - Man-months
- KDSI - Delivered source lines of code in thousands
- EM - Effort multipliers corresponding to the 15 predictor variables.

Development time as a project input parameter would have significant effect on the SLIM generated estimate but possibly no effect on the COCOMO generated estimate. Variance due to model bias will exist between the SLIM estimate and the COCOMO estimate for any project that uses development time as a project input parameter.

Equation 9 modifies the SLIM equation to solve for the effort variable (K).

$$K = Ss^{3}Ck^{-3}Td^{-4} \quad (9)$$

In both the COCOMO equation (Equation 8) and the modified SLIM equation (Equation 9), there is an exponential relationship between project size and required development effort. However, the exponent values are not equal and the result will be that different estimates of total required effort will be generated by each model based on project size as an input factor.

The conclusion to this example is that model bias will cause variance between estimates developed by different models. PM's must be knowledgeable about the model used in developing individual estimates that are submitted to the program office. A clear understanding of potential model biases that can be generated by automated models will help to explain why variance exists between estimates generated by different automated cost models.

2. Incomplete Lifecycle Coverage by Automated Models

Various automated cost models do not all provide the same coverage of the phases of the software development lifecycle as delineated in DOD-STD-2167A. Different models may cover all or only parts of the entire lifecycle. It is important for the PM to know what part of the development lifecycle is covered by the models that are producing the total project estimate because models that emphasize different lifecycle phases may produce different estimates for the same project.

Table 5 presents the previously discussed automated cost models and the respective lifecycle range that they cover [Ref. 18:p .19]. These models were not designed using a standard definition of lifecycle phases therefore the table will describe each model in its own terminology consistent with the documentation published by each model developer.

TABLE 5. SOFTWARE DEVELOPMENT LIFECYCLE RANGE COVERED BY EACH COST MODEL

MODEL	LIFECYCLE RANGE COVERED
SLIM	Feasibility Study to Full Operational Capability plus Operations and Maintenance
PRICE-S	Software Design through Test and Integration plus Operational Support of user specified length
JENSEN SYSTEM-4	Requirements Definition up to Development Test and Evaluation plus 15 years of Operational Support
SPQR/20	Planning through Integration/Test plus 5 years of Operational Support
COCOMO	Plans and Requirements Phase to Operations and Maintenance Phase
SASET	Systems Software Requirements Analysis to Systems Integration and Testing plus Maintenance

Managers need to be aware that different choices in automated models will result in different lifecycle phases being covered or emphasized in the cost estimate. Varying lifecycle coverage will effect the values of the total project cost estimate and cause variance between estimates developed by different models.

The methods used by the organizations developing the cost estimates will affect the outcome of the total project estimate. Model biases and incomplete lifecycle coverage may result when independent estimators use different models based on dissimilar methods to generate a cost estimate. In the next section, the inputs to the cost models will be analyzed to determine how inconsistent assumptions between independent estimators about input values affect the total project estimate.

B. INCONSISTENT ASSUMPTIONS BETWEEN INDEPENDENT ESTIMATORS ON THE PROJECT INPUT PARAMETERS

The most frequently cited contribution to variance between independent estimates is inconsistent assumptions between estimators on the elements comprising a software project while assigned to the same project. Independent estimators will inevitably have different opinions on the software project inputs that are used to generate a cost estimate. These differences will result in variance in the estimate for the entire project.

Certain inputs will be more critical than others and act as primary causes of project estimate variance. Unless estimates are being generated from the same initial assumptions on inputs, the estimates will not have a meaningful relationship to each other. The manager will not be able to validate the estimates interrelationship and will not have the confidence that multiple estimates should provide.

PM's, cost analysts, and engineers in the software development industry were interviewed to determine which input parameters had the greatest effect on the total estimate when inconsistent assumptions were made concerning their values. The consensus of opinion is that there are four areas of inputs for a software development program that act as primary causes of project cost variance [Refs. 8, 17, 19, 20, 21, 22, 23, 24, 25]:

1. Size estimate of the project
2. Software system characteristics
3. Software development complexities
4. Database selection for project calibration factors

These four areas will be analyzed to determine to what extent inconsistent assumptions of their values influence the outcome of the estimation process.

1. Size Estimate of the Project

The estimated size of a software project is a key factor in the total cost estimate. Most models use size as a primary variable in the estimation equations whether the model generates size or size is input directly. For example, the equation for effort in Intermediate COCOMO (Embedded mode) is (Equation 10):

$$MM = 2.8(KDSI)^{1.20}(\Pi EM) \qquad (10)$$

where:
- MM - Man-months
- KDSI - Delivered source lines of code in thousands
- EM - Effort multipliers corresponding to the 15 predictor variables.

In COCOMO, size is a primary input variable. To isolate the effect size has on total effort, the partial derivative with respect to the size variable (KDSI) was derived from the effort equation (Equation 10) and presented as Equation 11.

$$MM' = 2.8(1.20(KDSI)^{0.20})(\Pi EM) \quad (11)$$

This new effort equation indicates that the marginal change in effort is an exponential function of size. Percent change in size as an input will yield a greater percent change in the effort estimate. When considering the Organic and Semi-detached development modes, the relationship continue to hold true. However, the two simpler development modes, with the smaller exponent in the equation, do not impose as large of a marginal change in effort for change in size as the highly constrained Embedded mode.

Sensitivity analysis of input size versus output effort was conducted with the SASET model to empirically validate the relationship derived from the COCOMO model. From the BASE model, size was increased 25 percent to 148,125 LOC and decreased 25 percent to 88,875 LOC. As a result of these changes in size (as the input variable), the total effort estimates changed as well (as the output variable). The percent change in output effort can be calculated and compared to the percent change in the input size (exactly 25 percent). In both the cases of size being increased and size being decreased, the total effort estimate changed exactly 25 percent. This indicates that within the SASET model, there is a 1.00:1.00 relationship in percent change between changes in effort and changes in size.

This validates the COCOMO model. In both the SASET and COCOMO models, changes in the estimate of size as an input parameter resulted in significant changes in the estimate of total effort for the project. The relationship between percent change in size and percent change in effort is not identical for both models, however, some variation between the models is anticipated due to the differences in estimation methods within the models. It is likely that SASET does not weight size as heavily as COCOMO because SASET has a greater number of parametric inputs that additionally classify software project characteristics. The net effect of changes in size on the total estimated project effort is significant and comparable for both models.

Managers need to be aware that any inaccuracies in the initial project size estimate will translate into a significant error in the estimate of total effort required to develop the project. Table 6 presents the results of the sensitivity analysis of size as a cause of variance.

TABLE 6. PERCENT CHANGE IN THE ESTIMATED SIZE OF THE SOFTWARE PROJECT AND THE EFFECT ON THE TOTAL PROJECT EFFORT REQUIREMENT

Size (LOC) BASE +/- 25%	Effort required to complete the project man-months)	Percent change in required Effort	Percent change in Effort versus Percent change in Size
118,500 (BASE)	4023.04	N/A	N/A
148,125 (+25%)	5028.80	25.00%	1.00:1.00
88,875 (-25%)	3017.28	25.00%	1.00:1.00

2. Software System Characteristics

Software system characteristics are those factors that describe the project from a high-level perspective. System characteristics include class of software, primary software language, software type, and code condition. The valuation by the estimator of each of these characteristics will have an effect on the output value of the estimate.

In Intermediate COCOMO, the system characteristics are embedded in the effort multipliers (EMs). To isolate the effect of the EMs on the total effort estimate, the first order partial derivative for each of the 15 EM's is required. Each EM is multiplied against the nominal equation and will affect the marginal change in effort in the same manner. Equation 12 is the partial derivative of the COCOMO effort equation with respect to the variable EM.

$$\frac{MM}{dEM_i} = 2.8(KDSI)^{1.2}\left(\prod_{i=1}^{15}\frac{EM_i}{dEM_i}\right) \qquad (12)$$

In the first order derivative, there is a multiplicative relationship between the EM variable and the output of the equation in terms of effort. Any percent change in an EM will have an equivalent percent change in total effort required for the project, a 1.00:1.00 relationship. It should be noted, however, that this is true only if one predictor variable is changed at a time. Any combination of variables changed will result in the product of the percent

changes of each EM applied to the nominal equation exponentially vice multiplicatively and therefore a greater than 1.00:1.00 relationship will exist.

Sensitivity analysis was run on SASET to empirically determine the relationship between system characteristics and total effort. Four characteristics will be analyzed: class of software, primary software language, software type and code condition.

Table 7 displays the percent change in estimated effort for a change in each of the software characteristics. The input choices to the BASE model were changed to reflect some of the options the estimator has when developing an estimate with SASET. When using SASET, subjective choices made by the estimator on inputs will have significant effect on the total effort estimate. For example, a misclassification of software type may result in a percent change in estimated effort of over 74 percent.

In the SASET model, three of the four system characteristics analyzed affected the total project effort estimate by over 20 percent by changing the input selection available to the estimator. The analysis of the COCOMO equations and the sensitivity analysis of SASET both indicate that high level system characteristics of a software project are causes of variance between independent estimates. Careful consideration must be given to the subjective choices for software system characteristics when developing a cost

TABLE 7. THE EFFECT OF CHANGES IN SYSTEM CHARACTERISTICS ON THE TOTAL EFFORT REQUIRED TO COMPLETE THE DEVELOPMENT PROJECT

Class of Software	Effort	Percent change
Avionics (BASE)	4023.04	-
Unmanned flight	6705.06	66.67%
Ship/Submarine	3017.28	25.00%
Primary Software Language		
Ada (BASE) low experience	4023.04	-
Assembly	3965.29	1.44%
HOL experienced Ada	3686.16	8.37%
Software Type		
Systems (BASE)	4023.04	-
Application	2316.29	42.42%
Support	1036.24	74.24%
Code Condition "new-modified-rehost"		
100%-0%-0% (BASE)	4023.04	-
75-25-0	3751.48	6.75%
75-0-25	3117.85	22.50%
50-25-25	2846.30	29.25%

estimate as these characteristics have a significant effect on the outcome of total estimate.

3. Software Development Complexities

Software development complexities represent the working details of the software development process and include factors such as software requirements, timing and criticality, and programmer experience. The subjective value assigned to each of these complexities by the estimator will affect the output of the total cost estimate.

In COCOMO, these complexities are embedded in the EM's in the same manner as the system characteristics. Software development complexities have a multiplicative effect on the total effort estimate that results in a 1:00:1:00 relationship between percent change in the input complexities and the percent change in the output effort estimate. Any significant change in assigned complexity values will have significant effect on the cost estimate.

SASET incorporated 20 complexity factors into the model that affect the effort estimate. The model also provides an additional 12 integration factors that apply to projects that consist of multiple CSCI's. The BASE project is designed with only one CSCI, therefore, the integration factors default to 1 and have no effect as multipliers.

The complexity factors have four settings available for the user: "Very Complex", "Complex", "Average", and "Simple". Average is the default value and has a multiplier value of 1.0. The other selections have multiplier values as set by the calibration database and tend to range from 0.85 to 1.05. Sensitivity analysis was conducted on SASET with the BASE project and the results are compiled in Table 8. For each complexity factor, SASET was run for all four settings of complexity. "EFFORT" is the new computed estimate for required effort in man-months and "PERCENT CHANGE" is the percent change from the Base value to the new value for effort. The shift between Simple to Average settings created

TABLE 8. THE EFFECT OF CHANGES IN INDIVIDUAL COMPLEXITY FACTORS ON THE TOTAL EFFORT REQUIRED TO COMPLETE THE DEVELOPMENT PROJECT.

COMPLEXITY FACTORS	Very Complex	Complex	Average	Simple
System Requirements	4363 / 8.5%	4123 / 2.5%	BASE	3822 / -4.9%
Software Requirements	4223 / 5.0%	4123 / 2.5%	BASE	3822 / -4.9%
Software Documentation	4123 / 2.5%	4063 / 1.0%	BASE	3722 / -7.5%
Travel Requirements	4494 /11.7%	4423 / 9.9%	4376 / -8.8%	BASE
Man Interaction	4081 / 1.5%	BASE	3983 / -1.0%	3784 / -5.9
Timing and Criticality	4081 / 1.5%	BASE	3983 / -1.0%	3784 / -5.9
Software Testability	4081 / 1.5%	BASE	3983 / -1.0%	3689 / -8.3%
Hardware Constraints	4081 / 1.5%	BASE	3983 / -1.0%	3784 / -5.9
Hardware Experience	4123 / 2.5%	4063 / 1.0%	BASE	3722 / -7.5%
Software Experience	4123 / 2.5%	4063 / 1.0%	BASE	3722 / -7.5%
Software Interfaces	4123 / 2.5%	4063 / 1.0%	BASE	3722 / -7.5%
Development Facilities	4081 / 1.5%	BASE	3983 / -1.0%	3689 / -8.3%
Development versus Host Systems	4123 / 2.5%	4063 / 1.0%	BASE	3722 / -7.5%
Technology Impacts	4123 / 2.5%	4083 / 1.5%	BASE	3722 / -7.5%
Off the Shelf Software	4083 / 1.5%	4063 / 1.0%	BASE	3722 / -7.5%
Development Team	4123 / 2.5%	4063 / 1.0%	BASE	3722 / -7.5%
Embedded Systems	4123 / 2.5%	4083 / 1.5%	BASE	3722 / -7.5%
Development Tools	4083 / 1.5%	4063 / 1.0%	BASE	3722 / -7.5%
Personnel Resources	4083 / 1.5%	4063 / 1.0%	BASE	3722 / -7.5%
Programming Language	4083 / 1.5%	.4063 / 1.0%	BASE	3722 / -7.5%

the largest variance of any of the other single setting shifts. For example, 16 of the 18 complexity factors resulted in a percent change in effort required of over 7.5 percent. The average for all the other shifts between complexity settings was approximately 2.5 percent. There is a trend of less variation in the shifts between the median settings (Complex and Average) than the shifts to the extreme settings (Complex and Very Complex or Average and Simple).

Software development complexities have an impact on the total effort estimate generated with an automated model. With COCOMO, the effect is that the percent change in the input value is applied multiplicatively to the total effort. This results in a 1.00:1.00 relationship between percent change in the inputs to percent change in the output. SASET empirically tends to decrease the estimate of effort required to a greater degree when relaxing complexity inputs than it increases the estimate of effort when complexity inputs become more constrained from nominal. There also appears to be greater variation in the required total project effort when the extreme complexity settings are chosen (very complex and simple) than when the middle settings are chosen (complex and average).

4. Database Selection

The use of an appropriate historical database for parametric estimation is important for accurate software cost estimation. The database is used to calculate the calibration

coefficients that weight the various input choices offered to the user in automated models. The database is also used in sizing models to store actual function task values that may be analogous to function tasks in the project being estimated.

The database needs to be large enough to provide reasonable confidence to the user that the full spectrum of the predicted project range is covered. The database may also be tailored to the specific type of project under development, such as avionics or business application, to emphasize specific traits of the project type into the calibration coefficients. It is appropriate, however, to have some variety in project type within the database because many diverse projects share common characteristics such as language type and programmer experience.

The accumulation of information for databases has been handled poorly in the past. Data specific to software engineering is difficult to find and to extract from higher levels within the WBS. PM's have recently recognized the problem and have specified in the contract Request for Proposal (RFP) that the contractor shall specifically delineate software costs. In the future, it should be easier to obtain software development costs to update databases with the most current software development project information.

For the BASE model, the SASET default calibration database (TIER.CAL) was used to calibrate the weight coefficients used to calculate the estimate. For sensitivity

analysis, each calibration weight was altered to simulate that the calibration weights were re-calculated based upon different database information. The calibration weights, typically ranging from 0.85 to 1.05, were increased by a factor of 0.01 and 0.03 simulating an increasingly complex and constrained project database. The calibration factors were then decreased 0.01 and 0.03 simulating a more simple and flexible project database. The effect on the estimate of project effort was significant and ranged from 16.41 to 65.37 percent. The complete results of the sensitivity analysis are graphically presented in Figure 8.

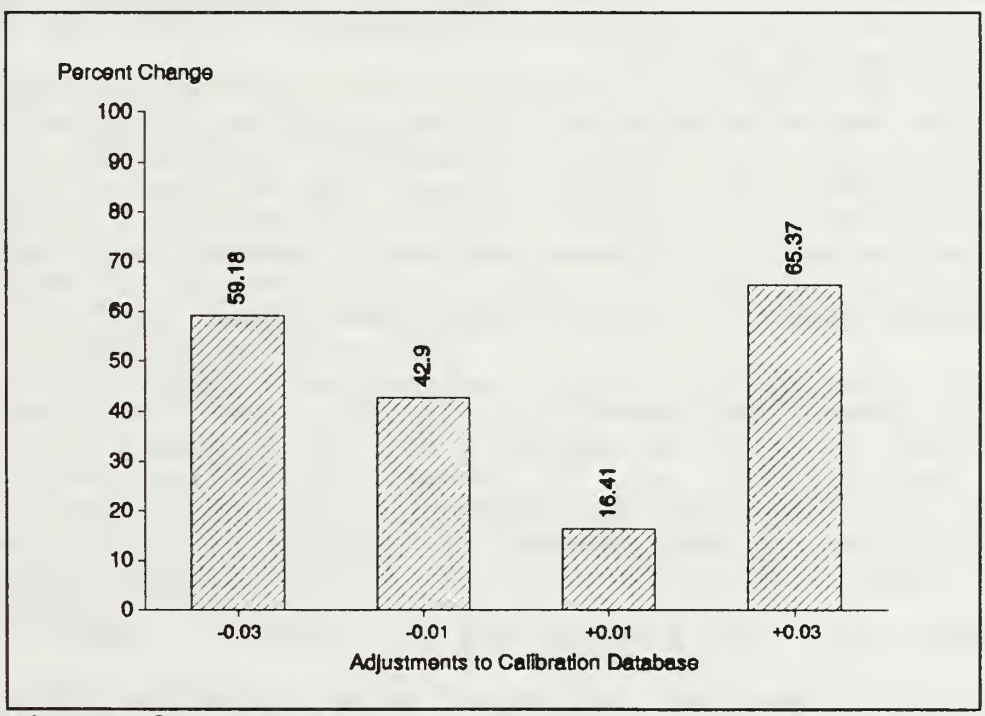

Figure 8. The effect of changes in calibration factors as determined by the database on the total effort required to complete the development project.

The results indicate that there are large changes in the estimate of project effort even with small calibration changes in the calibration weights as determined by the selection of the historical database.

C. SUMMARY

Software development cost estimates generated from independent estimators on the same project will normally have some variance between the estimates of total effort requirements to complete the project. The manager who has a clear understanding of the factors influencing these variances will be better equipped to reconcile the variances and produce an estimate that accurately represents the resources required to develop the project.

The choice of methods for developing the software project cost estimate will directly impact the outcome of the total project estimate. Different estimation methods will emphasize different aspects of the software development process and may result in model bias and/or incomplete lifecycle coverage.

Model bias is an emphasis by the automated cost model given to certain favored elements of the estimation process. This emphasis skews the outcome of the estimate towards the elements that are favored and may include: effort verses schedule tradeoffs, estimated size (LOC) as a primary independent variable, the weighing of certain project complexity factors, or the intended application of the software project. Estimates submitted to the PM that have

been generated on different cost models may have variance between the estimates for total project cost because of model bias.

Incomplete lifecycle coverage may result because different cost models may cover all or only parts of the entire lifecycle. It is important for the PM to know what part of the development lifecycle is covered by the models that are producing the total project estimate because models that emphasize different lifecycle phases may produce different estimates for the same project.

Different assumptions between estimators on the parameters comprising a software project is the most frequently cited driver of variance. Independent estimators may develop a cost estimate for a project with inconsistent assumptions depending on the estimators information, experience, and personal prejudices. The result is that variances will exist between estimates of the same project when more than one estimator or group of estimators are used.

There are four areas of a software development program identified as primary causes of the total project cost variance due to inconsistent assumptions on project input parameters. These four areas are:

1. Size estimate of the project
2. Software system characteristics
3. Software development complexities
4. Database selection for project calibration factors

Unless estimates are being generated from the consistent initial assumptions, independent estimates will not have a meaningful relationship to each other. The manager will not be able to validate the estimates interrelationship and will not have the confidence that multiple estimates should provide.

The choice of methods used to generate the estimate and the consistency of the assumptions concerning the inputs to the cost estimate will both have significant effect on the total project cost estimate. In Chapter V, methods for reducing variance between independently developed estimates will be discussed. Techniques for reconciling estimates generated by independent sources, procedures for reducing differences in initial assumptions, and DOD organizations and tools to assist a program manager will be introduced to improve the process of software development cost estimation.

V. VARIANCE REDUCTION BETWEEN INDEPENDENT COST ESTIMATES

There are many factors that can cause a variance between estimates developed by independent estimators. Managers charged with budget responsibility must be able to identify the causes of the variances and extract the valuable information from each estimate. In spite of the fact that software development cost estimation is not an exact science, the manager needs to be able to make an absolute decision for budget submission. This chapter presents ways to reduce the variance and the effect of variance between software development cost estimates generated by independent estimators. Variance reduction is discussed in four distinct areas: combining estimates generated from separate sources, reducing inconsistencies in input parameters between independent cost estimation organizations, organizations to assist the PM in the estimation process, and the use of the Software Engineering Institute Capability Maturity Model.

A. COMBINING ESTIMATES GENERATED FROM DIFFERENT SOURCES

It is important to have more than one estimate of software development cost to insulate a manager from the effect of model bias and/or incomplete lifecycle coverage. This is particularly true in light of the industry's history of significant cost overruns and schedule slippage for software

development projects. Software cost models and estimators have different strengths and weaknesses. The difficult question for the manager is how to combine the different cost estimates to harness the strengths of each estimate and to cast aside the misleading information.

Nearly all cost estimation organizations recognize the need for multiple estimates generated by independent estimators. However, there is very little attention given to the methods or techniques used to combine the estimates into one estimate that best represents the development project. This section of the chapter analyzes various mathematical techniques to determine a method that effectively combines the independent estimates.

The result of the analysis will be a weighted equation that combines the estimates from each cost model and produces a new Combined Estimated Value (CEV). The CEV is closer in value to the actual value of the project than any of the input models could have generated independently. The weighted equation will be in the form of Equation 13.

$$CEV = \sum_{i=1}^{m} \sum_{j=1}^{n} W_i E_{ij} \qquad (13)$$

Where:

CEV = the new combined estimated value of all the models
i = the independent automated cost model
j = the projects in the sample
m = the total number of cost models
n = the total number of projects in the sample
W = the weight of cost model i
E = the estimated value from cost model i for project j

A sample of past project data is required to formulate the weighted equation by relating the models' estimates to the actual project values. Each individual cost model will calculate an estimate for each project in the sample. The individual project estimates will then be combined with the other model estimates of the projects to form the CEV. The sum of the weights of the cost models are not required to equal one, and considering the industry's history of underestimating software costs, it is likely that the weights will sum to a value greater than one.

The data for the formulation of the techniques in this study were compiled from the avionics project section of the SASET database. Included in the database were the actual values of the effort in man-months (MM) of the software development projects. Software project input parameters given within the database were line of code (LOC), software complexity factors, software class, software project type, and host hardware information. The projects range from 1,300 to 325,000 LOC, low to high complexity factors, six software class choices, three software project types, and a variety of host hardware systems including the F-16, F-111 and X-15 advanced aircraft. A complete breakdown of all the data points and their parameters is included in Appendix A.

Given the input parameters of the projects in the database, estimates for each project were calculated with two automated models. SASET and the COSTMODL version of COCOMO

were used for the analysis because both these models are user friendly, compatible to the standard DOD PC, available at no cost to DOD users, and relatively accurate. Between the two models, the entire lifecycle of the software development project is fully covered. These two estimates for each project will be combined in various ways to determine the CEV that most closely matches the actual project values.

A measure for comparing the results of the CEV to the actual project value is needed. Two different statistical formulae were considered to compare the CEV to the actual values: Mean Squared Error (MSE) and Mean Absolute Percent Error (MAPE) (Equations 14 and 15).

$$MSE \ldots\ldots \sum_{j=1}^{n} \frac{(A_j - CEV_j)^2}{n} \qquad (14)$$

$$MAPE \ldots\ldots \frac{100}{n} \sum_{j=1}^{n} \frac{|A_j - CEV_j|}{A_j} \qquad (15)$$

where:
A = actual value for effort of project j
CEV = the new combined estimated value of all the models
j = the projects in the database
n = the total number of projects in the database

MAPE was selected for the project analysis because of the wide dispersion within the size (LOC) of the projects in the database. Since MAPE is a percent error, it is possible to

make relevant comparisons of error between projects of different sizes.

MSE was also calculated for each technique analyzed. MSE inherently weights the error prediction in favor of the larger (more costly) projects. In this study, with a large range of project size in the database (1,300 - 325,000 LOC), MSE is influenced to such a significant degree by errors in the larger projects that MSE loses significance for the smaller projects.

For example, for all the techniques analyzed, the best prediction of error by MSE was 8,916 MM's. However, only one of the 31 projects analyzed had a actual value of effort greater than 8,916 MM's. This large prediction of MSE is the result of squaring the errors in the largest projects and then dividing through by a relatively small sample number of 31. The wide dispersion in project size within this database results in predictions of MSE having little value to evaluate techniques to effectively combine the estimates from the different cost models. It is the intent of this study to provide information to the manager without preference towards larger projects and therefore MAPE will be used as the measure of error.

Eleven different algorithms were employed to attempt to identify the optimal procedure to combine the individual cost model estimates. The optimal combined estimate will result in a minimum MAPE, provide the benefits of multi-model cost

development estimate, and be more accurate on the average than any single model independently. The different choices of techniques analyzed, the equations used, and the results in terms of MAPE and MSE are presented in Table 9.

TABLE 9. TECHNIQUES FOR COMBINING INDEPENDENT COST MODEL RESULTS INTO A SINGLE OPTIMAL ESTIMATE

TECHNIQUE	EQUATION USED TO COMBINE MODELS	MSE	MAPE (%)
Linear analysis			
COCOMO Model	C	21,477	20.84
SASET Model	S	11,417	17.07
Average of both models	CEV = (C+S)/2	14,304	16.10
Regression (COCOMO vs Actual Values)	CEV = 1.26(C)	11,449	24.60
Regression (SASET vs Actual Values)	CEV = 1.09(S)	9,834	16.51
Multiple regression on both models	CEV = 0.48(C)+0.68(S)	8,916	19.84
Linear program solution for both models	CEV = 0.43(C)+0.59(S)	12,476	15.81
Logarithmic analysis			
Regress (COCOMO vs Actual Values)	logCEV = 1.03(logC)	12,191	19.57
Regress (SASET vs Actual Values)	logCEV = 0.99(logS)	11,417	16.51
Multiple regression on both models	logCEV = 0.39(logC) + 0.62(logS)	9,811	16.54
Linear program solution for both models	logCEV = 0.1465(logC) + .8535(logS)	11,821	16.25

The results of this analysis will specifically apply to this database only. However, this database is representative of many avionics databases and the results of this analysis could be directly applied to other avionics projects with a

minimum of expected deviation from the results in this study.[Ref. 24] For projects outside the scope of avionics, the techniques applied here can be reapplied to the specific software development areas of interest to determine a CEV equation for that area.

The first two predictions for CEV were the individual estimates of each cost model. The independent estimates calculated from each model were compared directly to the actual values of the project in terms of MAPE with relatively accurate results. COCOMO was analyzed first and resulted in a MAPE of 20.84 percent; SASET was then analyzed and resulted in a MAPE of 17.07 percent.

The arithmetic average of the independent estimates generated by the two cost models was calculated as a technique to combine the estimates. MAPE for the average estimate was 16.10 percent, slightly lower than either individual model estimate indicating that a CEV derived from an arithmetic average is preferable to either individual model estimate. Figure 9 presents the relationships between the COCOMO estimate, the SASET estimate, and the average of the two in terms of MAPE.

Linear regression was used to develop equations correlating the estimates generated by the cost models to the actual project values. Linear regression will solve for the equation of the line that best fits the data points. That line equation can then be used to make predictions relating

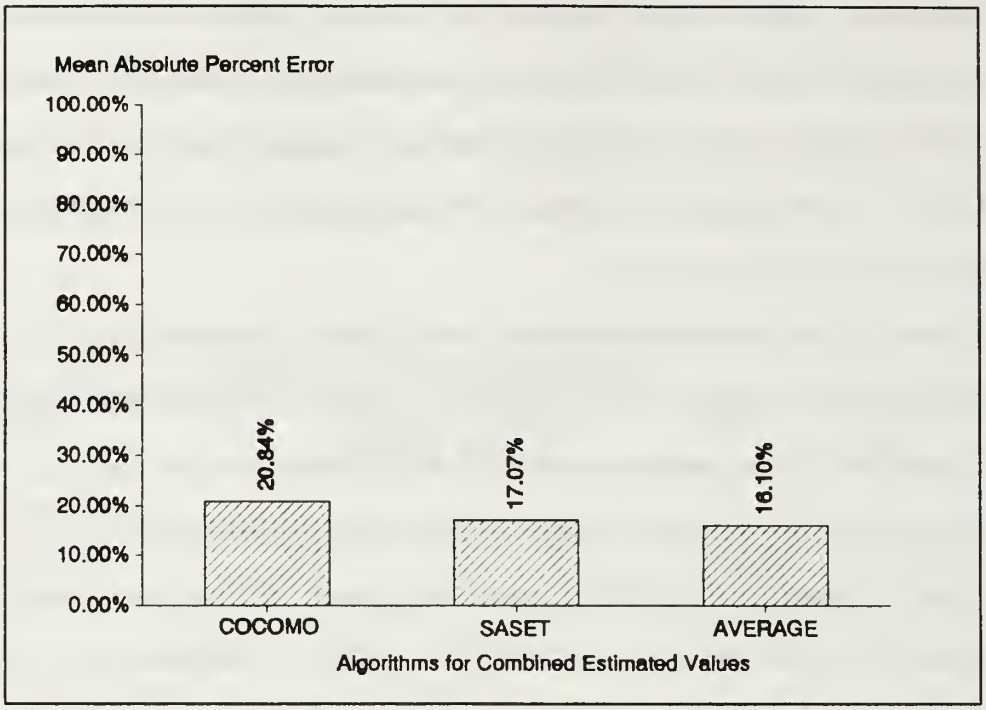

Figure 9. MAPE's for COCOMO, SASET, and their Average used as techniques to estimate actual project values

the estimates of the cost models to the expected value of the actual project. The regression analysis was calculated to force the Y axis intercept to zero. Although some accuracy might be sacrificed in the overall best fit by forcing the Y intercept to zero, it was necessary to keep the percent error of the smaller LOC projects in the relevant range.

The regression equation correlating the COCOMO estimates of the individual projects in the database to the actual project values is Equation 16. The regression plot is presented in Figure 10. Regression with the COCOMO

$$CEV = 1.255(C) \quad (16)$$

where:
CEV = the estimate of the actual project value
C = the estimate developed by the COCOMO model

model results in a MAPE of 24.60 percent. This is significantly higher than all the previously derived MAPEs and will not be considered a good technique to generate an estimate of the software development project cost.

The regression equation relating the estimates generated by the SASET model to the actual values was then developed (Equation 17). The regression plot for the SASET model is presented as Figure 11. Regression with the SASET model results in

$$CEV = 1.09(S) \quad (17)$$

where:
CEV = the estimate of the actual project value
S = the estimate developed by the SASET model

a MAPE of 19.61 percent. This is higher than the previously developed MAPE for the model average and therefore will not be further considered as an alternative equation to optimally correlate a cost model estimate to the project's actual values.

A multiple regression equation was formulated to correlate the estimates of the COCOMO model and the SASET model to the actual project values (Equation 18). The sum of the weights applied to each model is greater than one, indicating that

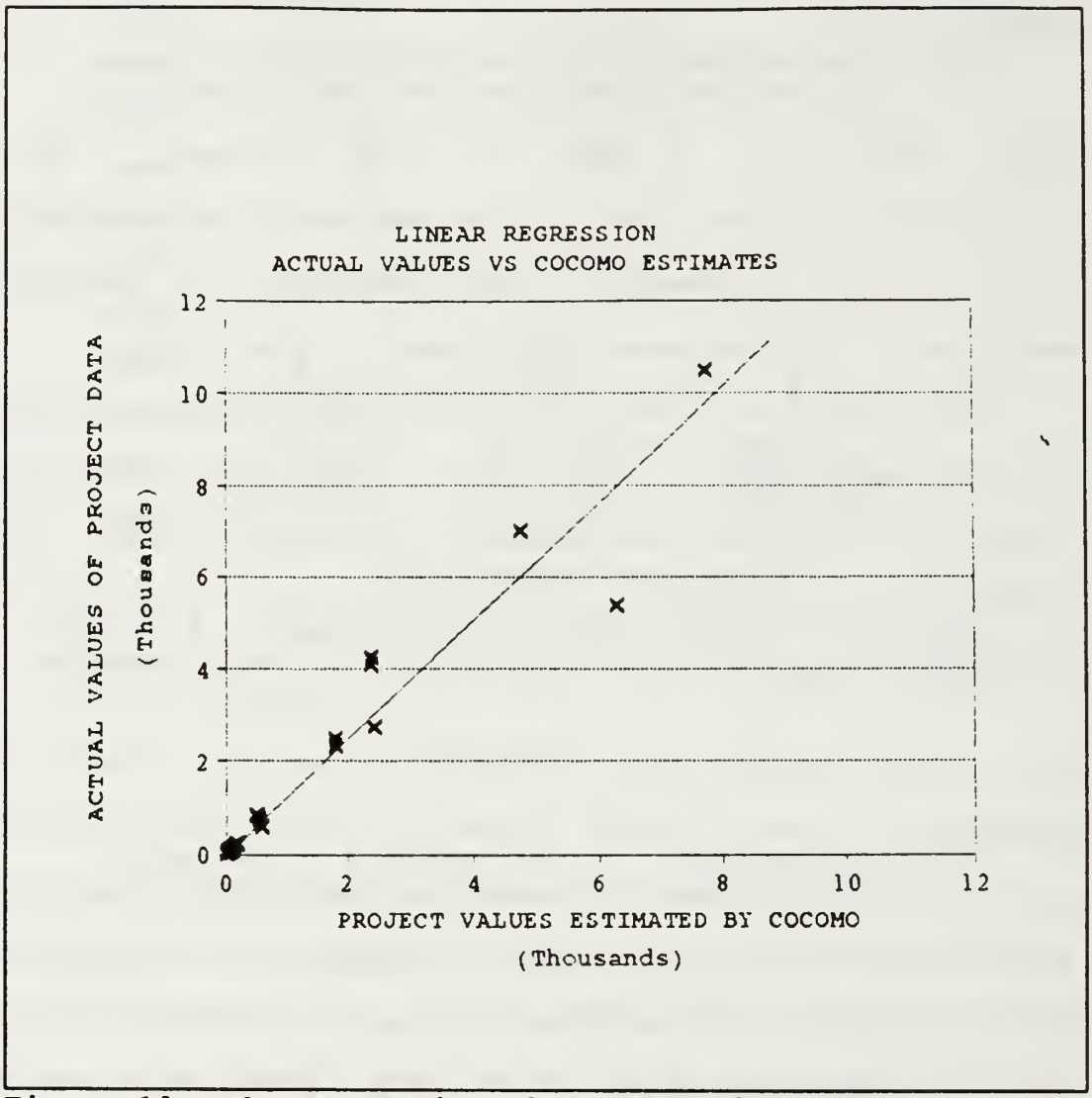

Figure 10. The regression plot presenting the correlation between the COCOMO estimates and the actual project values

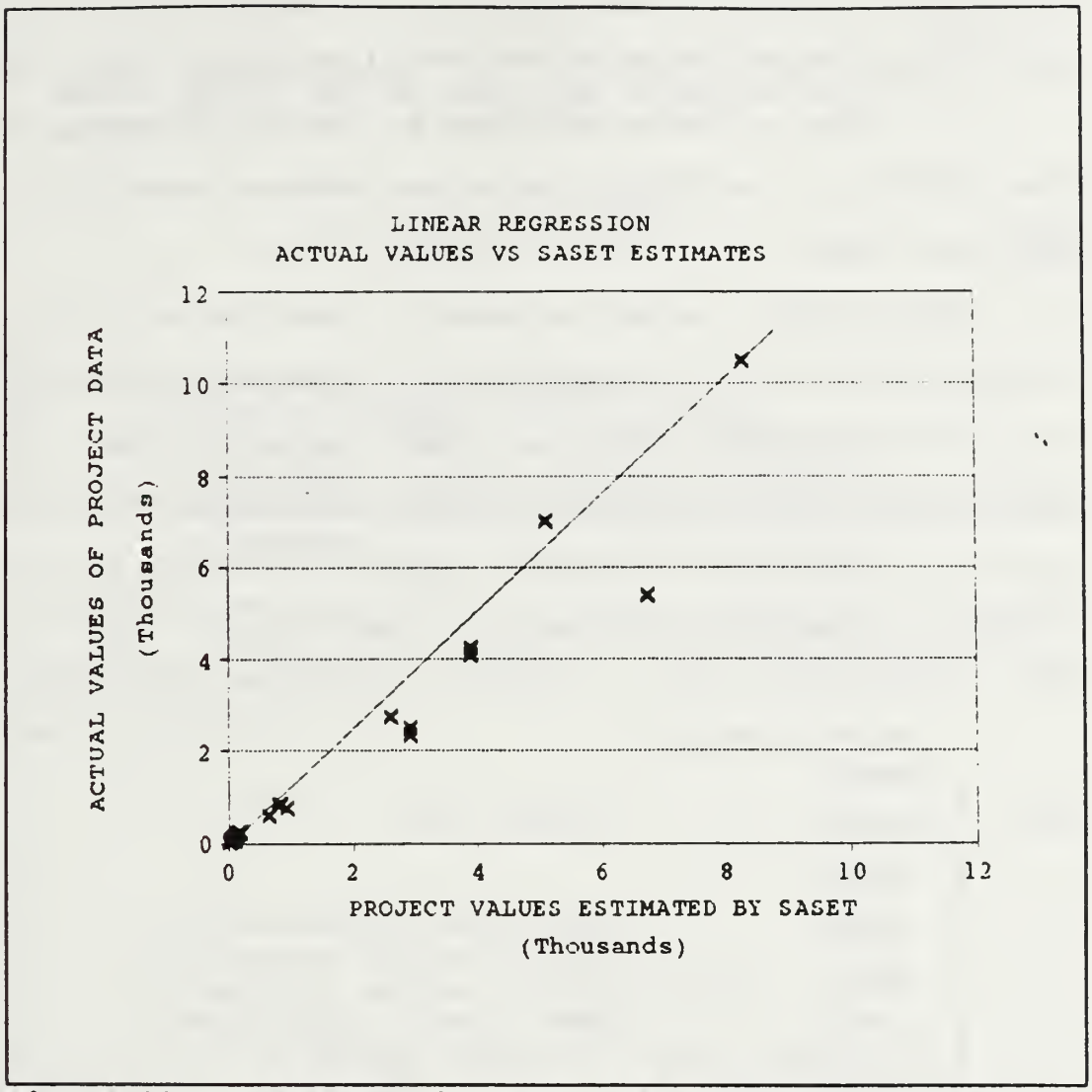

Figure 11. The regression plot presenting the correlation between the SASET estimates and the actual project values

$$CEV=0.48(C)+0.68(S) \quad (18)$$

where:

CEV = the estimate of the actual project value
C = the estimate developed by the COCOMO model
S = the estimate developed by the SASET model

the COCOMO and the SASET models had underestimated the actual project value.

The multiple regression equation resulted in a CEV for the project data that had a MAPE of 19.84 percent with respect to the actual project values. This is higher than the MAPE resulting from the average of the two models and therefore does not represent the optimal equation relating the two cost models to the project's actual values (Figure 12).

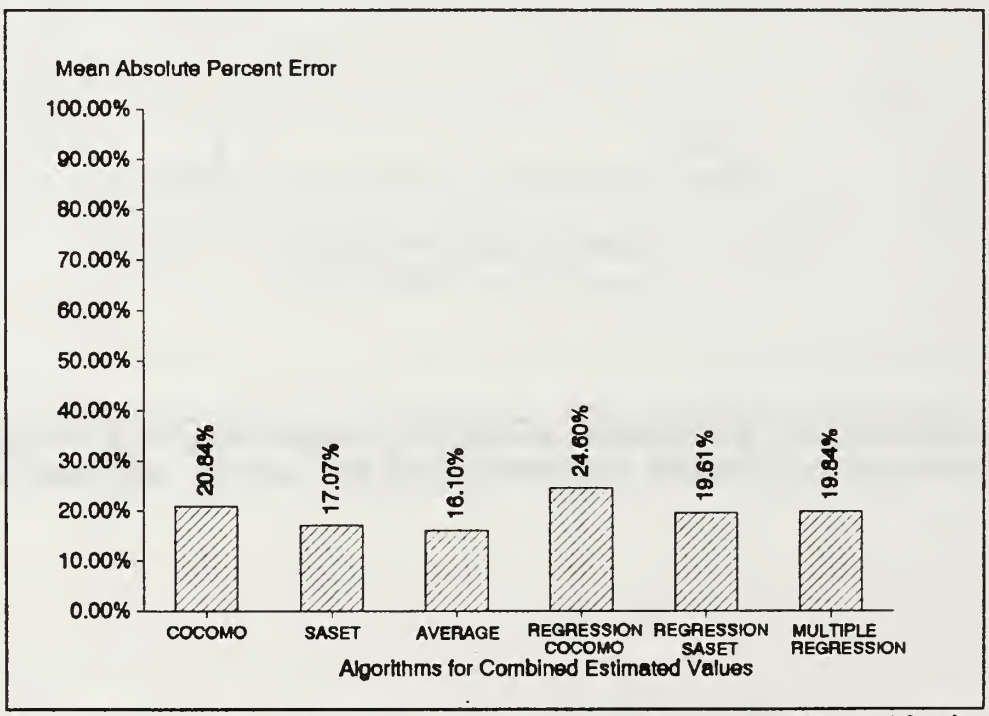

Figure 12. MAPE's for COCOMO, SASET, their Average, and Regression equations used as techniques to estimate actual project values.

Linear programming was then used to try to find an optimal solution to fit the estimated values produced by COCOMO and SASET to the actual values. The objective was to minimize the MAPE (Equation 19).

$$\text{Min} \frac{100}{m} \sum_{j=1}^{n} \frac{|A_j - CEV_j|}{A_j} \tag{19}$$

where:

j	=	a project in the database
m	=	the total number of cost models
n	=	the number of projects in the database
A	=	the actual project value of project j
CEV	=	the combined estimate of the actual project value of project i

The CEV is determined by the linear program as a function of the weight factors applied to the individual model estimates (Equation 20).

$$CEV_j = \sum_{i=1}^{m} W_m E_m \tag{20}$$

where:

i	=	the automated cost model
j	=	a project in the database
m	=	the total number of cost models
W	=	the weight of cost model i
E	=	the estimated value from cost model i

The objective function of the model was formulated to minimize the MAPE similar to Equation 19. The constraints to the model were the values for the weights of Equation 20 that satisfied each of the 31 projects in the database. The sum of the weights for each cost model were not constrained to equal one to allow the linear program to compensate for underestimates by the individual cost models of the actual

project values. All objective and constraint equations used in the linear program are included in Appendix B. The program was run on the Linear, Interactive, and Discrete Optimizer (LINDO), an automated model for solving linear programs.

The relationship generated by the linear program to relate the best combination of single model estimates to the actual values is Equation 21.

$$CEV=0.43(C)+0.59(S) \quad (21)$$

where:

CEV = the combined estimate of the actual project value
C = the estimate developed by the COCOMO model
S = the estimate developed by the SASET model

The linear program solution was designed to combine independent model estimates to produce a CEV that has minimum deviation from the actual values of the projects in the database. The measure of that deviation, the MAPE, was 15.81 percent for the linear program. This linear programming solution has produced the optimal equation to this point in terms of minimizing MAPE and employing the benefits of a multiple model estimate. Figure 13 presents the techniques discussed thus far and their respective values of MAPE.

After linear analysis, all data values were transformed to logarithms to test for a better exponential relationship between the estimates by the cost models and the actual project values. The results of the exponential analysis were not better than the results from the original linear program (Figure 14). Additionally, the extra work and training

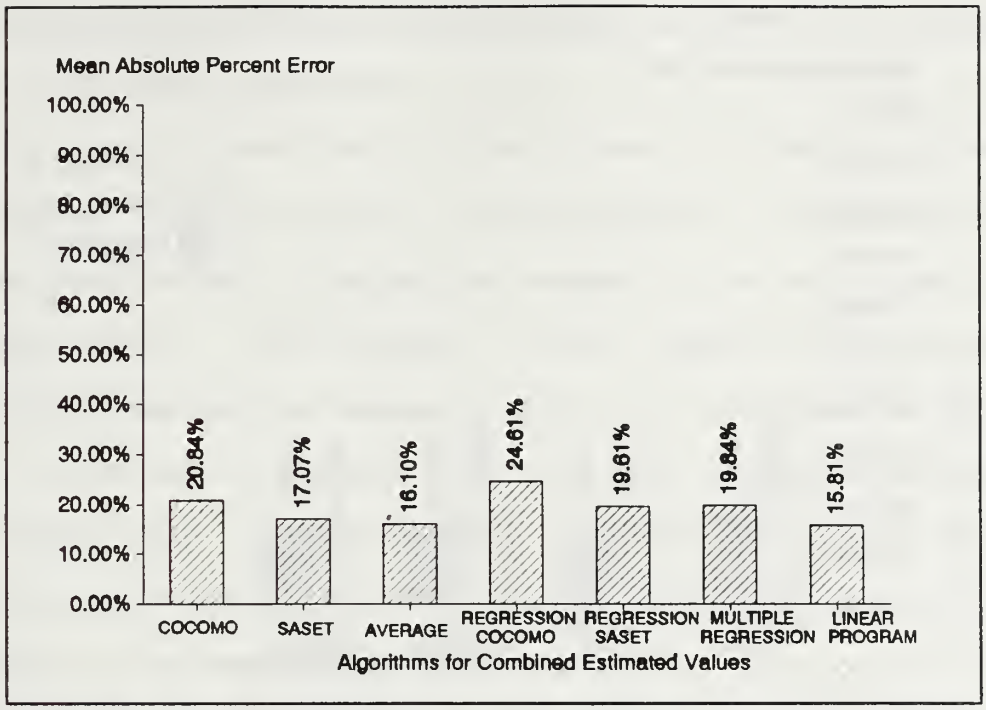

Figure 13. MAPE's for COCOMO, SASET, their Average, Regression Equations, and the Linear Program used as techniques to estimate actual project values.

required to generate the logarithmic analysis makes the logarithmic analysis less desirable for the PM than the linear analysis techniques. There was not a logarithmic algorithm that outperformed the linear program solution in terms of minimizing MAPE and providing a multiple model estimate.

The conclusion from this analysis is that the best relationship between the estimated values from COCOMO and SASET is calculated using a linear program representing the estimates of two models. For this particular avionics database, the equation of the optimal relationship is Equation 21.

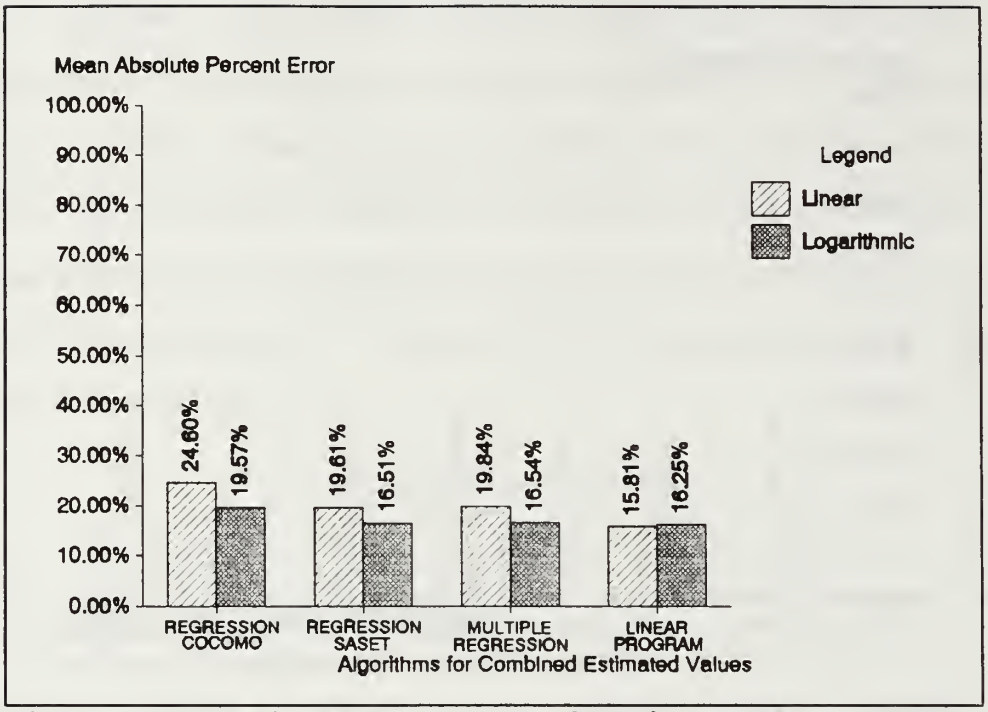

Figure 14. Linear and Logarithmic MAPE's for the Regression Equations and the Linear Program used as techniques to estimate actual project values.

This weighted equation represents the optimal CEV that minimizes MAPE between the combined estimate and the actual project values. By using multiple models, the equation also reduces the uncertainty for the PM caused by model bias and possible incomplete lifecycle coverage. The equation can be used by any manager who is estimating an avionics software project and has access to DOD provided COSTMODL (COCOMO) and SASET. In a broad sense, the techniques can be adapted to any project that has sufficient actual information available to perform the mathematical techniques to generate a new weighted equation tailored to that type of project. The weighted equation will provide a more accurate estimate of the software

development project than either COCOMO or SASET could generate as an individual estimate.

A second alternative to the linear program would be the arithmetic average of the two estimates. Generating the linear program may be beyond the scope of the resources of the program office. The average, in the case of this project, provided relatively accurate results as evidenced by the low MAPE. However, an average does not have the capability to compensate for estimates that typically under-cost the software development project. The average of the independent estimates received by the program office should be used to combine the estimates into a superior composite estimate if the resources to formulate a linear program are not available.

B. REDUCING INCONSISTENCIES BETWEEN INPUT PARAMETERS OF INDEPENDENT COST ESTIMATION ORGANIZATIONS

Reducing the inconsistencies between the input parameters used by estimation organizations will reduce the variance between the output estimates of these organizations. It is difficult for a manager to make legitimate comparisons between independent estimates if the estimates have been generated from a different set of input parameters. Within a program office, the PM must know what assumptions were made in generating the cost estimates before he/she can be reasonably confident of their validity. A series of questions were developed to aid the PM in determining those assumptions.

1. **What Cost Estimation Methodology was used?**

The choice of cost estimation methodology will affect the results of the estimate and the manner in which the PM views the estimate. Was the estimate produced using parametric, analogy, top-down, bottom-up, expert judgment, or price-to-win methods. If automated models were used, what were the equations representing the relationships between input and output values. How were these relationships derived in terms of:

a. What database was utilized? For example, an avionics software project should be developed from a model that includes avionics data in the database as in the BASE project.

b. How good of a statistical fit existed in the relationship between input and output values of the model. Fit can be measured by correlation coefficient (R), coefficient of determination (R squared) and standard error of estimate (SEE).

The better the PM understands the methods used to develop the estimates, the better equipped he/she will be to make comparisons between the estimates and reconcile the differences between them.

2. **How is a Line of Code defined?**

As discussed in Chapter II, there are many different ways to define what a line of code is. The more common methods are:

1. Count only executable lines.

2. Count executable and data definition lines.
3. Count executable, data definition, and comment lines.
4. Count executable, data definition, comment, and job control lines.
5. Count lines as physical lines on an input screen.
6. Count lines as terminated by logical delimiters.
7. Count lines as a combination of the above methods.

Different definitions will change the initial size estimate used to generate the cost estimate. Since size is a key variable in many cost models, variance in the size estimate drastically affects the estimate of total cost. The PM should specifically understand the working definition of LOC for each independent estimate before comparisons between the estimates are made.

3. **What are the Software System Characteristics?**

The PM should know the assumptions made about the software system characteristics. System characteristics form the high level structure of the estimate and small differences in these characteristics may have significant effect on the outcome of the cost estimate. For example, software code may be classified as either new, modified, re-use, or any combination of the three. As the combination changes within a project, the estimate of cost will vary significantly. Other examples of software system characteristics include class of software (i.e. avionics or manned flight), primary

software language (i.e. Ada, Fortran, or assembly), and software type (i.e. system, application, or support).

4. **What are the assumptions made regarding software project complexity?**

Various assumptions must be made by the cost estimator on project complexities as the estimate is being developed. These assumptions will affect the outcome of the total estimate of cost since many models apply these complexities as linear multiplier calibration weights. The PM should have a realistic understanding of what assumptions were made and what values the calibration weights for each complexity factor were assigned. This will help the PM to make informed decisions in comparing cost estimates because he/she will know specifically how each complexity factor was weighted and how it affected the total estimate.

5. **Has the contractor already included into the estimate a factor for cost growth?**

A PM can and should expect cost growth during the project acquisition cycle. However, the estimate received from the independent estimator should not be "padded" but should accurately reflect the most likely estimate of project cost based on the best information available. The PM must know what growth factors are included in all the independent estimates so a valid comparison of the estimates can be made. It is also important that the PM know what, if any, growth factors are already embedded into the estimate so that he or

she can to avoid duplicating the expected growth factor and creating a redundant cost requirement.

These five questions will serve the PM to help reduce the variance in software project estimation by reducing the inconsistencies between the input parameters used by the different estimation organizations. To facilitate an open exchange of information regarding these questions, the PM should state in the RFP that the contractor will be required to provide information in response to the above questions.

C. ORGANIZATIONS TO ASSIST THE PROGRAM MANAGER

There are organizations within the U.S. Navy that can provide assistance to the PM in terms of expert knowledge, material resources, and/or an independent estimate. These organizations and the services they provide can reduce the effect of variance between project estimates and improving the overall quality of the cost estimation process by enhancing the PM's ability to understand the more delicate aspects of the software development cost estimation process. These organizations have personnel specifically trained for cost analysis and represent the leading edge in the cost estimation field.

1. System Command Organizations

The System Commands (SYSCOM's) have central cost estimating and analysis groups that have been established to support their respective PMs with responsive, quality cost

estimation services. These groups have the capability to provide a variety of cost estimation services either within their own staffing or through their support activities.[Ref. 26] These estimation/analysis groups provide the following functions:

1. Cost Estimating for:
   a. Programming and Budget Submission
   b. Development and production programs
   c. Operating and Support costs
   d. Contracting and contract modifications

2. Cost Analysis for:
   a. Source selection and evaluation
   b. Contract negotiation
   c. Tracking and update of cost database
   d. Economic analysis and program evaluations
   e. Cost effectiveness studies

3. Other Services
   a. Development of costing models and cost estimating technique
   b. Maintenance of historical cost database
   c. Cost estimating consultation, special reports, and presentation
   d. Cost estimating and analysis training
   e. Focal point for obtaining outside cost estimating services

The cost estimating and analysis groups within each SYSCOM are staffed with highly specialized professional costing personnel. There is a cost estimating and analysis group within each SYSCOM. The following list presents the cost estimating division for each SYSCOM.

1. AIR-524 Naval Air Systems Command
2. SEA-017 Naval Sea Systems Command
3. SPAWAR-02 Space and Naval Warfare Systems Command

The PM should utilize these groups to the fullest extent possible in all cost estimating aspects of the program.

2. Naval Center for Cost Analysis

The Naval Center for Cost Analysis (NCA) is an organization created to provide an independent cost analysis capability for the Secretary of the Navy (SECNAV) and the DOD Cost Analysis Improvement Group (CAIG). The NCA will generate an independent cost estimate (ICE) to validate the program office's estimate before funds are granted for major programs to ensure credible lifecycle cost estimates are generated to support PPBS and system acquisition.

The NCA reports to SECNAV within a separate chain of command than the SYSCOM's to facilitate the important requirement that the two organizations remain independent from each other. This independent operating procedure facilitates the NCA's effort to submit an estimate that can be used to validate the program office's estimate without any pressure from the program office. The NCA reports to SECNAV directly through the Assistant Secretary of the Navy for Financial Management (ASN(FM)), while the SYSCOMs report to SECNAV through the CNO's office.

However, the NCA and the program office often have some interaction prior to formally submitting their individual cost estimates to the DOD level CAIG. This interaction serves to allow the groups to come to an informal agreement on

the software development cost estimate, usually within 10 percent of each other. This does not defeat the purpose of the NCA as they still provide an independent estimate, although most the "negotiating" between the NCA and the program office occurs prior to the CAIG.

In addition to preparing Independent Cost Estimates (ICE's), the NCA is charged with the responsibility to:

1. Guide, direct, and strengthen cost analysis within the Department of the NAVY (DON).
2. Perform miscellaneous tasks such as economic and special studies.

Although separated by the formal chain of command, the NCA can provide services to the PM in the form of expert advice and material resources. The NCA is a highly professional organization that specializes in the field of cost analysis.

The cost analysis organizations within the SYSCOMs and the NCA can provide expert assistance to the program office in the cost estimation field. The PM should rely on the specialized knowledge and experience the personnel in these organizations can offer for cost estimation of software development.

D. SOFTWARE ENGINEERING INSTITUTE CAPABILITY MATURITY MODEL

The Software Engineering Institute (SEI) of Carnegie-Mellon University has developed the Capability Maturity Model

(CMM) that is designed for evaluating the software development process of a government contractor. SEI is a Federally funded research institute that was established in December of 1984 to address the problems that the DOD was experiencing with software development. SEI has developed a comprehensive project, the Software Process Measurement Project (SPMP), that is designed to assess the capabilities of a software development organization. The CMM is a tool within the SPMP that the PM can use to determine the software development process maturity of a contractor.

Within the CMM, there are five levels of process maturity for an organization [Ref. 27]. Software process maturity indicates both the richness of an organization's software process and the consistency with which the process is applied in projects throughout the organization. The CMM is based on the premise that maturity is an indicator of the capability and reliability of the software development process. The five levels of process maturity are presented in Figure 15.

1. **Initial**

The software development process is generally ad hoc, poorly controlled, and redefined for each new project. Project outcomes are characterized by large cost and schedule overruns and a low level of software quality.

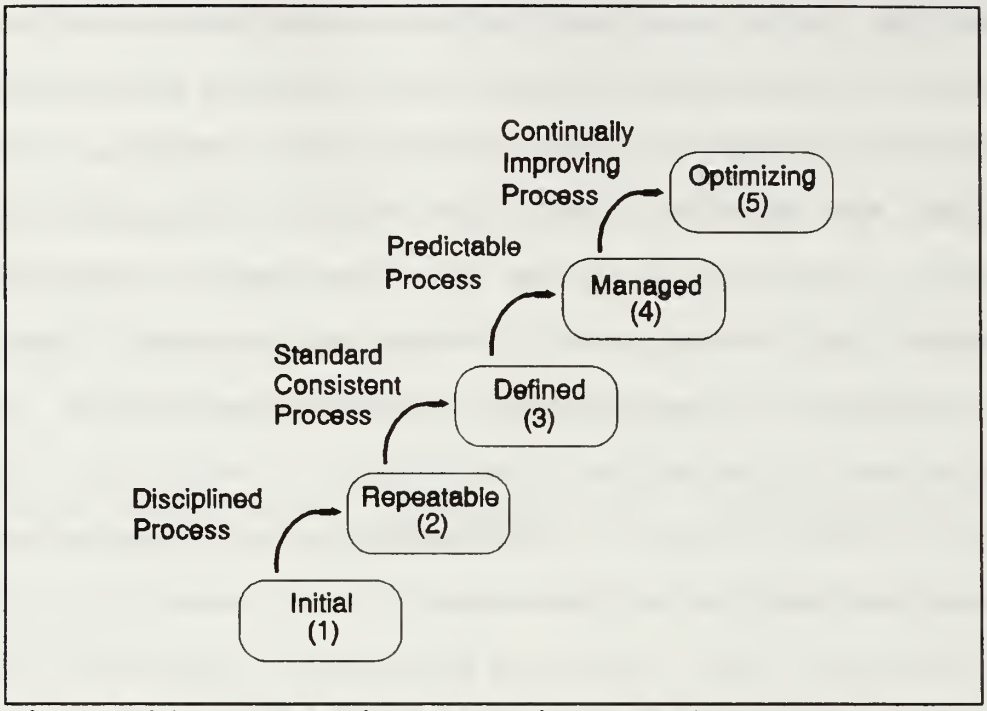

Figure 15. The Five Maturity Levels of The CMM.

2. Repeatable

Management controls are put in place to standardize the company's procedures and software quality assurance. The development process is now in control for projects that do not widely vary from the norm. Any significant changes in product type, however, may destroy the relevance of the controls and return the process back to ad hoc.

3. Defined

Process standards are institutionalized first within the organization and then tailored to the project type. Project results are more predictable across a broader range of projects. The process, however, is still defined using only qualitative measurements of performance.

4. Managed

Quantitative quality and productivity tools are placed into the development process. Goals are specifically set for each defined step in the process; success or failure in meeting goals is recorded. High predictability is achieved not only for each project as a whole, but also for each step along the project's process.

5. Optimizing

Quantitative data from each process step is used to pinpoint process weaknesses and bottlenecks. Causes of errors are determined and analyzed with action taken to prevent similar errors in the future. Process improvement is now embedded in project development and the process has strong capabilities across a wide spectrum of projects.

The PM can determine the maturity level of a prospective contractor by requesting that a contractor complete the Maturity Questionnaire part of the CMM. The Maturity Questionnaire consists of 120 questions covering 18 Key Process Areas (KPAs) consisting of a total of 343 Key Processes (KPs). Most contractors have offered no resistance to the requirement of the questionnaire and many defense contractors have already completed the process maturity evaluation. To date, a relatively effective and efficient contractor has typical rated a "Defined" on the evaluation [Ref. 20][Ref. 21]. It is the goal of DOD to use the SEI to

further educate and train organizations in areas of process improvement to reach an "Optimizing" level and therefore maximize the effectiveness and efficiency of the software development process.

The CMM provides a tool for the PM to assess the relative strengths of a prospective contractor in regards to their software development process. The higher the credibility of a contractor, the less risk is present of cost and schedule errors in the estimate. To date, most contractors rate 1 - 3 on the CMM. The contractor with a more mature software development process should have a reliable estimating process that is subject to minimal variance due to errors and inaccuracies.

E. SUMMARY

Variance between independent software development cost estimates may reduce the manager's ability to confidently generate a composite budget quality estimate. The manager must understand why variance exists and how to reconcile it.

The use of multiple cost models increases the managers confidence that the combined estimate is not adversely affected by either different lifecycle phase emphasis or model biases. However, methods for combining the multiple estimates have not been comprehensively developed in the academic arena or in actual practice. This study has analyzed five general approaches encompassing 11 different algorithms to combine

multiple cost models that result in minimal expected error between the estimate and the actual values. A linear programming solution was empirically determined to be the optimal algorithm to produce a CEV for an avionic system software development project using SASET and COCOMO (COSTMODL version). Reducing the differences in input parameters between independent estimation organizations will also reduce variance between the estimates generated from each organization. This chapter presented five questions that the PM can use to reduce the significant drivers of variance between the contractor and the program office. These five questions are:

1. What cost estimation methodology was used?
2. How is LOC defined?
3. What are the software system characteristics?
4. What are the assumptions made regarding software project complexity?
5. Has the contractor already included into the estimate a factor for cost growth?

The PM has support organizations available to assist in the cost estimation process. The Cost Analysis Division within each SYSCOM and the NCA are available to help the PM with cost estimation, cost analysis, and other related issues. The PM should rely on these organizations as they are specifically designed for the cost analysis role and are

staffed with highly specialized and knowledgeable professionals.

SEI has developed a Capability Maturity Model for the DOD that the PM can use to assess the maturity of the estimation organization's software development process. A 120 question survey is completed by the contractor to provide the PM information required to rate the contractor's software development as either Initial, Repeatable, Defined, Managed, or Optimizing. A contractor with a mature software development process will have a reliable cost estimating operation as well.

How to reduce variance and the effect of variance between independent estimators is an important tool to the manager charged with budget responsibility. The manager must be able to sift through the available cost estimate information and make an absolute decision for budget submission. The tools presented in this chapter will assist the PM in controlling variance between independent estimates and to gain control of the software development cost estimation process.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Cost estimation of software development has been a critical problem for the DOD over the last 35 years. Software development costs have grown to exceed hardware development costs within computer resource projects as technology has advanced from analog to digital equipment. Current technology projects use software intensive computer resources for command, control, and communication of highly sophisticated electronic systems. Errors or inaccuracies in the cost estimate of the software development project in the early stages of the acquisition cycle will result in significant variance between the estimate and the actual costs of the final product.

Variance will also exist between the software project estimates that the PM receives from the contractor, from the ICE, and from within his/her own organization. However, the PM is responsible for combining the independent estimates and submitting a composite estimate that best reflects the most likely cost of the software project. The PM needs to have a clear understanding of why variance exists between these estimates and how to reduce the variance to confidently reconcile the individual estimates into a valid combined

estimate. This study has presented the reasons why variance exists between independently generated estimates and how the variance can be reduced.

1. Why Variance Exists Between Independently Developed Cost Estimates

Why variance exists between independent estimates can be broadly categorized into two major areas:

1. Methods used to develop the estimates.
2. Inconsistent assumptions between independent estimators on project input parameters.

Both these areas were closely analyzed to determine what specific factors cause variance between the independently generated estimates for the same software development project.

a. Methods used to develop the estimates

The choice of methods for developing the software project cost estimate will directly impact the outcome of the total project estimate. Different estimation methods will emphasize different aspects of the software development process and may result in model bias and/or incomplete lifecycle coverage.

Model bias is an emphasis by the automated cost model given to certain favored elements of the estimation process. This emphasis skews the outcome of the estimate towards the elements that are favored and may include: effort verses schedule tradeoffs, estimated size (e.g. lines of code)

as a primary independent variable, the weighing of certain project complexity factors, or the intended application of the software project. Estimates submitted to the PM that have been generated on different cost models may have variance between the project estimates because of model bias.

Incomplete lifecycle coverage may result because different cost models may cover all or only parts of the entire lifecycle. It is important for the PM to know what part of the development lifecycle is covered by the models that are producing the project estimates because models that emphasize different lifecycle phases may produce different estimates for the same project.

b. Inconsistent assumptions between independent estimators on project input parameters

Different assumptions between estimators on the parameters comprising a software project is the most frequently cited cause of variance. Independent estimators may develop a cost estimate for a project with different assumptions on inputs depending on the estimators information, experience, and personal prejudices. The result is that variances will exist between estimates of the same project when more than one estimator or group of estimators are used.

There are four areas of a software development program identified as primary causes of the variance between estimates due to inconsistent assumptions on project input parameters. These four areas are:

1. Size estimate of the project
2. Software system characteristics
3. Software development complexities
4. Database selection for project calibration factors

Unless estimates are being generated from consistent initial assumptions, independent estimates will not have a meaningful relationship to each other. The manager will not be able to validate the estimates' interrelationships and will not have the confidence that multiple estimates should provide. Methods to reduce variance between estimates are required to decrease the negative effect of the variances on the cost estimation process.

2. How To Reduce Variance Between Cost Estimates Generated By Independent Estimators

Understanding how to reduce variance and the effect of variance between estimates generated by independent estimators is an important capability for the manager charged with budget responsibility. The manager must be able to sift through the available cost estimate information and make an absolute decision for budget submission. Most cost estimators recognize the need for multiple estimates to harness the strengths of each individual estimate and to cast aside the misleading information. However, methodology for combining multiple estimates into one superior composite estimate is a largely undefined task within program offices.

a. *Combining estimates generated from different sources*

The study analyzed techniques to combine individual estimates into a superior composite estimate. Estimates for analyses were generated using the SASET and COCOMO (COSTMODL version) models based on real project data from the avionics database within the SASET model. The objective was to determine the optimal relationship that combines the SASET and COCOMO estimates to minimize MAPE between the combined estimate and actual project values.

Five general techniques were used to determine the best estimate to predict the actual project values: single model estimates, arithmetic averages of the estimates, linear regression, linear programming, and logarithmic analysis of the linear techniques. The optimal relationship between the estimated values from the two models resulted from linear programming (Equation 21).

$$CEV = 0.43(C) + 0.59(S) \quad (21)$$

where:

CEV = the estimate of the actual project value
C = the estimate developed by the COCOMO model
S = the estimate developed by the SASET model

Equation 21 will provide a combined estimate, CEV, that is more accurate than a single model estimate by either SASET or COCOMO (COSTMODL) for this particular database. The sum of the weights of the two models in the equation is

greater than one indicating that each model tends to underestimate the total project cost. The avionics database in this project is representative of current avionics databases and therefore this equation can also be used by any manager who is estimating an avionics software project and has access to DOD provided COSTMODL (COCOMO) and SASET.

A second alternative to the linear program would be the arithmetic average of the two estimates. The average provides the confidence gained by coverage of more than one cost model and may produce more accurate estimates than any of the individual input estimates (as was the case in the project example). However, an average does not have the capability to compensate for estimates that typically under-cost the software development project. The average of the independent estimates received by the program office should be used to combine the estimates into a superior composite estimate if the resources to formulate a linear program are not available.

In a broad sense, the techniques applied to develop a CEV for this database can be adapted to any project. The difficulty is finding sufficient data to perform the mathematical techniques to generate a new weighted equation specifically tailored to that type of project.

b. Reducing inconsistencies between input parameters of independent cost estimation organizations

Variance between estimates produced by independent organizations can also be reduced by decreasing the

inconsistencies in input parameters used by the independent organizations to generate the estimates. Inconsistencies in input parameters can be reduced by an improved exchange of information between the estimation organizations. The study has presented five key questions the PM can ask of the independent estimation organizations to establish a common base of information to evaluate the individual estimates from. Those questions are:

1. What cost estimation methodology was used?
2. How is LOC defined?
3. What are the software system characteristics?
4. What are the assumptions made regarding software project complexity?
5. Has the contractor already included into the estimate a factor for cost growth?

With these questions answered, the PM is in a better position to make decisions regarding each of the individual estimates.

c. Organizations to assist the Program Manager

The PM within the DON has organizations available to assist in decisions regarding cost estimation. Each SYSCOM has a cost analysis division staffed with highly specialized professional cost analysts that can support the program office in cost estimating, cost analysis, and other services. Another organization available to assist the PM is the Naval Center for Cost Analysis. The NCA is an independent DON

organization that can also support the PM in the same areas as the SYSCOM divisions in addition to it's responsibility for providing an ICE for SECNAV's use. The PM should rely on the cost estimation groups within the SYSCOMs and the NCA as these organizations are specifically designed for the cost analysis role and are staffed with highly specialized and knowledgeable professionals.

d. Software Engineering Institute Capability Maturity Model

Outside of DON organizations, the PM has available a powerful tool developed by the DOD sponsored Software Engineering Institute called the Capability Maturity Model. The CMM is useful to assess the software development process maturity of a cost estimation organization through an extensive 120 question survey filled out by the estimation organization. Software process maturity indicates both the richness of an organization's software process and the consistency with which the process is applied in projects throughout the organization.

The PM can use the CMM to help establish the credibility of a contractor and reduce the uncertainty in the software development cost estimation process. The higher the software development process maturity of an organization, the less risk is present of cost and schedule overruns. The contractor with a mature process should have a reliable

estimating process that is subject to minimal variance due to errors and inaccuracies.

Understanding why there is variance in cost estimates between independent cost estimators and understanding how to reduce this variance will greatly improve the software development estimation process for the PM. The manager entrusted with budget responsibility will be better equipped to produce a cost estimate of the development project that accurately reflects the most likely requirement of resources necessary to complete the project.

B. RECOMMENDATIONS FOR THE PROGRAM MANAGER

Based on the research and conclusions of the study, decision rules were formulated for the PM as recommendations for reducing variance between independent estimates of software cost. These decision rules will reduce the uncertainty in the estimate process and aid the PM in developing an accurate estimate of total project cost. The applicable section where each decision rule is supported within the study is provided with each rule to facilitate referencing each decision rule for more details.

1. Use multiple cost models

Use multiple cost models to reduce model bias and to ensure complete lifecycle coverage of the development process. SASET and the COSTMODL version of COCOMO are recommended for DOD users because both these models are user friendly,

compatible to the standard DOD PC, available at no cost to DOD users, relatively accurate, and between them provide complete coverage of the project development lifecycle. [Chapter V., section A.]

2. Use a CEV weighted equation

Use a CEV weighted equation formulated by the process developed in this study to combine independent cost estimates into a superior composite estimate. The equation formulated in this study, Equation 21, can be applied to avionics software development projects since it was developed from an avionics project database. The average of the independent estimates is the second alternative to the CEV weighted equation.[Chapter V., section A.]

3. Use the provided list of questions for defining initial assumptions

The PM should use the provided list of questions to reduce inconsistencies between assumptions of the initial input parameters to the cost models by the independent cost estimators.[Chapter V., section B.]

1. What cost estimation methodology was used?
2. How is LOC defined?
3. What is the condition of software code?
4. What are the assumptions made regarding software project complexity?
5. Is there embedded growth factors already incorporated into the cost estimate?

4. Use existing DON resources for assistance

Use the organizations within the DON to assist the PM in terms of expert knowledge, material resources, and independent estimates. Multiple estimates and opinions validate each other and reduce uncertainty. Existing organizations include the cost analyst divisions in the SYSCOMs (AIR-524, SEA-017,and SPAWAR-02) and the NCA.[Chapter V., section C.]

5. Use the Capability Maturity Model

Use the Capability Maturity Model provided by SEI to evaluate the maturity of the software development process of a contractor or other independent estimator. A more mature development process in synonymous with a more capable and reliable process and therefore allows greater confidence in the final project estimate for the PM.[Chapter V., section D.]

In summary, cost estimation of software development projects has been an elusive task that has given managers problems since the early days of computer resources 35 years ago. Cost overruns and schedule slippage has been the trademark vice the exception in the software development field. However, managers are still responsible for submitting an estimate for the software development project that represents the most likely cost of the project. The use of these five decision rules will assist the PM in gaining control over the software development cost estimation process

and producing a budget estimate that represents the most likely estimate of the total project cost.

C. AREAS FOR FURTHER RESEARCH

The study of software development is a enormous field and is rapidly growing as computer technology and software manufacturing resources continue to develop. Cost estimation of this volatile field is a complex task and must be continual researched to keep up with the advances in computer resources. The following subjects are recommended areas for further research to continue improving the software cost estimating process.

1. Apply the Combined Estimated Value equation to areas outside of avionics databases

The CEV process developed in this study was applied specifically to avionics databases. The CEV equation should be formulated to cover a wide variety of databases to create a set of equations that managers can employ to develop cost estimates for the full spectrum of software development projects.

2. Appraise the new RCA-Price model

RCA-Price has announced the release of a new cost model scheduled for July 1992. The new model is compatible to IBM PC's and requires a 80386 processor and a math co-processor to maximize the programs capabilities. The current RCA-Price models are available only as a time-share or as a

lease arrangement. The new model should be analyzed to determine the benefits it may offer to the DOD.

3. Software Engineering Institute

The Software Engineering Institute has been sponsored by DOD to address the problems DOD has been having in the software development field. SEI has developed a comprehensive project, the Software Process Measurement Project (SPMP), that is designed to assess the capabilities of a software development organization in the cost estimation field to improve the quality of the cost estimation process. Further research of the progress SEI has made and the continued dissemination of their results may improve the process of cost estimation within the DOD.

APPENDIX A.

BASE PROJECT PARAMETERS

Summary information on the 31 projects' parameters in the hypothetical avionics database "BASE."

Project	Complexity	Software Type	LOC	Effort(MM)
1.	LOW	SUPPORT	1300	12
2.	MEDIUM	SYSTEMS	1427	81
3.	MEDIUM	SUPPORT	1700	18
4.	HIGH	SYSTEMS	1733	12
5.	HIGH	APPLICATION	2331	98
6.	MEDIUM	SYSTEMS	2400	116
7.	MEDIUM	APPLICATION	2667	64
8.	LOW	SUPPORT	3000	28
9.	MEDIUM	APPLICATION	3000	100
10.	MEDIUM	SUPPORT	3300	30
11.	MEDIUM	APPLICATION	4067	93
12.	MEDIUM	APPLICATION	700	119
13.	MEDIUM	APPLICATION	4798	176
14.	MEDIUM	APPLICATION	5000	119
15.	HIGH	SUPPORT	6000	45
16.	MEDIUM	APPLICATION	6167	152
17.	LOW	SUPPORT	9865	94
18.	LOW	APPLICATION	10752	225
19.	MEDIUM	SUPPORT	15000	204
20.	HIGH	APPLICATION	21000	800
21.	HIGH	APPLICATION	22000	800
22.	HIGH	APPLICATION	24000	755
23.	MEDIUM	APPLICATION	26000	567
24.	HIGH	APPLICATION	75000	2334
25.	HIGH	APPLICATION	75000	2500
26.	HIGH	APPLICATION	100000	4250
27.	HIGH	APPLICATION	100000	4067
28.	MEDIUM	APPLICATION	102000	2750
29.	MEDIUM	APPLICATION	200000	7000
30.	MEDIUM	APPLICATION	263767	5365
31.	MEDIUM	APPLICATION	325000	10500

APPENDIX B.

LINEAR PROGRAM EQUATIONS

Minimizing Mean Absolute Percent Error with a Linear Program

Indices:
- i cost model i = 1,...,m
- j observation j = 1,...,n

Data:
- E sub ij = predicted value of model i on observation j
- A sub j = actual value of observation j

Decision variables:
- W sub i = weight of predictive model i
- Z sub j = the absolute value of the jth term

Desire to minimize:

$$\text{Min} \frac{100}{m} \sum_{j=1}^{n} \frac{|A_j - CEV_j|}{A_j}$$

where

$$CEV_j = \sum_{i=1}^{m} W_m E_m$$

Let Z sub j represent the absolute value of the jth term

$$\text{MIN} \frac{1}{m} \sum_{j=1}^{n} Z_j$$

Subject to:

$$Z_j \geq \sum_{i=1}^{m} W_j \frac{S_{ij}}{|A_j|} - 1$$

$$Z_j \geq 1 - \sum_{i=1}^{m} W_j \frac{S_{ij}}{|A_j|}$$

The comprehensive list of equations is as follows:
observations (1 - 31) were represented by (A - AH)

MINIMIZE:

```
ZA + ZB + ZC + ZD + ZE + ZF + ZG + ZH + ZI + ZJ + ZK +
ZL + ZN + ZO + ZP + ZQ + ZR + ZS + ZT + ZU + ZV + ZW +
ZX + ZY + ZZ + ZAA + ZAB + ZAC + ZAD + ZAE + ZAF + ZAG +
ZAH + ZAI
```

SUBJECT TO:

```
 1)  ZA - 1.03 WC - 0.93 WS >= -1
 2)  ZA + 1.03 WC + 0.93 WS >=  1
 3)  ZB - 0.64 WC - 0.78 WS >= -1
 4)  ZB + 0.64 WC + 0.78 WS >=  1
 5)  ZC - 1.13 WC - 1.08 WS >= -1
 6)  ZC + 1.13 WC + 1.08 WS >=  1
 7)  ZD - 0.57 WC - 0.93 WS >= -1
 8)  ZD + 0.57 WC + 0.93 WS >=  1
 9)  ZE - 0.94 WC - 1.05 WS >= -1
10)  ZE + 0.94 WC + 1.05 WS >=  1
11)  ZF - 0.75 WC - 0.92 WS >= -1
12)  ZF + 0.75 WC + 0.92 WS >=  1
13)  ZG - 0.00 WC - 1.07 WS >= -1
14)  ZG + 0.00 WC + 1.07 WS >=  1
15)  ZH - 1.01 WC - 0.92 WS >= -1
16)  ZH + 1.01 WC + 0.92 WS >=  1
17)  ZI - 0.72 WC - 0.77 WS >= -1
18)  ZI + 0.72 WC + 0.77 WS >=  1
19)  ZJ - 1.31 WC - 1.26 WS >= -1
20)  ZJ + 1.31 WC + 1.26 WS >=  1
21)  ZK - 1.01 WC - 1.12 WS >= -1
22)  ZK + 1.01 WC + 1.12 WS >=  1
23)  ZN - 0.94 WC - 1.01 WS >= -1
24)  ZN + 0.94 WC + 1.01 WS >=  1
25)  ZO - 0.65 WC - 0.70 WS >= -1
26)  ZO + 0.65 WC + 0.70 WS >=  1
27)  ZP - 0.00 WC - 1.08 WS >= -1
28)  ZP + 0.00 WC + 1.08 WS >=  1
29)  ZQ - 1.59 WC - 2.33 WS >= -1
30)  ZQ + 1.59 WC + 2.33 WS >=  1
31)  ZR - 0.97 WC - 1.04 WS >= -1
32)  ZR + 0.97 WC + 1.04 WS >=  1
33)  ZT - 0.99 WC - 0.90 WS >= -1
34)  ZT + 0.99 WC + 0.90 WS >=  1
35)  ZU - 0.90 WC - 0.91 WS >= -1
36)  ZU + 0.90 WC + 0.91 WS >=  1
37)  ZV - 0.88 WC - 0.84 WS >= -1
38)  ZV + 0.88 WC + 0.84 WS >=  1
39)  ZW - 0.63 WC - 1.03 WS >= -1
```

```
40) ZW + 0.63 WC + 1.03 WS >=  1
41) ZX - 0.66 WC - 1.08 WS >= -1
42) ZX + 0.66 WC + 1.08 WS >=  1
43) ZY - 0.76 WC - 1.24 WS >= -1
44) ZY + 0.76 WC + 1.24 WS >=  1
45) ZZ - 1.10 WC - 1.17 WS >= -1
46) ZZ + 1.10 WC + 1.17 WS >=  1
47) ZAA - 0.77 WC - 1.26 WS >= -1
48) ZAA + 0.77 WC + 1.26 WS >=  1
49) ZAB - 0.72 WC - 1.17 WS >= -1
50) ZAB + 0.72 WC + 1.17 WS >=  1
51) ZAC - 0.56 WC - 0.92 WS >= -1
52) ZAC + 0.56 WC + 0.92 WS >=  1
53) ZAD - 0.59 WC - 0.96 WS >= -1
54) ZAD + 0.59 WC + 0.95 WS >=  1
55) ZAE - 0.89 WC - 0.95 WS >= -1
56) ZAE + 0.89 WC + 0.95 WS >=  1
57) ZAF - 0.68 WC - 0.73 WS >= -1
58) ZAF + 0.68 WC + 0.73 WS >=  1
59) ZAG - 1.17 WC - 1.26 WS >= -1
60) ZAG + 1.17 WC + 1.26 WS >=  1
61) ZAH - 0.74 WC - 0.79 WS >= -1
62) ZAH + 0.74 WC + 0.79 WS >=  1
```

LIST OF REFERENCES

1. Boehm, B.W., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, 1981.

2. Shyman, S.R., and Blankenship, T.K., Improving Methods for Estimating Software Development Costs, Institute for Defense Analysis, June 1990.

3. Putnam, L.H. and Wolverton, R.W., Quantitative Management Software Cost Estimation, IEEE Computer Society, November 1977.

4. Putnam, L.H., Software Cost Estimating and Lifecycle Control, IEEE Computer Society, October 1980.

5. Brooks, F.P., The Mythical Man-Month, Addison-Wesley, December 1974.

6. Naval Center for Cost Analysis, Software Cost Estimation Course Guide, 1991.

7. Kile, R.L., A Process View of Software Estimation, Presented at the 7th International COCOMO User's Group Meeting, November 1991.

8. Interview between S. Gross, Naval Center for Cost Analysis, and the author, 10 February 92.

9. Martin Marietta Corp., Software Cost Estimation Study, by Silver, A.N., July 1988.

10. Bourdon, G.A., A Computerized Model for Estimating Software Life Cycle Costs, Electronic Systems Division, July 1977.

11. Telephone conversation between Dickerman, C., Quantitative Software Management, Inc., and the author, 01 April 1992.

12. Clark, G.A., Software Cost Estimation Models, Which One to Use?, Master's Thesis, Air Command and Staff College Air University, Maxwell Air Force Base, Alabama, April 1986.

13. Telephone conversation between Krisch, M., RCA-Price Systems, and the author, 01 April 1992.

14. Telephone conversation between Katz, P., Computer Economics, Inc., and the author, 01 April 1992.

15. Telephone conversation between Zimmeran, J., Software Productivity Research, Inc., and the author, 01 April 1992.

16. Staker, R.D., Strengths and Weaknesses of Four Software Cost Estimating Models, Student Report, Air University, Maxwell Air Force Base, Alabama, June 1990.

17. Telephone conversation betwen Anderberg, M.R., Operations Research Analyst for Assistant Secretary of Defense (Program Analysis and Evaluation), and the author, 28 January 1992.

18. Bailey, E.K., Frazier, T.P., and Bailey, J.W., A Descriptive Evaluation of Automated Software Cost-Estimation Models, Institute for Defense Analysis, October 1986.

19. Interview between Wilson, R., Naval Center for Cost Analysis, and the author, 10 February 1992.

20. Interview between Colvert, R.G., Captain, United States Navy, Program Manager for Air ASW Systems, Naval Air Systems Command, and the author, 13 February 1992.

21. Interview between Bell, J.K., Captain, United States Navy, Program Manager for S-3A/B/ES-3A, Naval Air Systems Command, and the author, 13 February 1992.

22. Interview between Cable, L., Commander, United States Navy, Assistant Program Manager for Systems and Engineering (SH-60B), Naval Air Systems Command, and the author, 12 February 1992.

23. Interview between Shrank, A., Naval Air Systems Command, and the author, 12 February 1992.

24. Interview between Frazier, T.P., Institute for Defense Analysis, and the author, 12 February 1992.

25. Interview between Herd, J., Deputy/Technical Director, Naval Center for Cost Analysis, and the author, 12 February 1992.

26. Interview between Cardarelli, J., Cost Analysis Division (Avenue), Naval Air Systems Command, and the author, 11 February 1992.

27. Software Engineering Institute, Technical Report CMU/SEI-91-TR-24, Capability Maturity Model for Software, by M.C. Paulk, W. Curtis, and M.B. Chrissis, August 1991.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Professor Joseph San Miguel, Code AS/Sm Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor Michael Sovereign, Code OR/Sm Department of Operations Research Naval Postgraduate School Monterey, California 93943-5000	1
5.	Director ATTN: Mr. S. Gross, Code NCA-7 Naval Center for Cost Analysis Pentagon, Room 4A538 Washington, District of Columbia 20350-1100	2
6.	Institute for Defense Analysis ATTN: Dr. Thomas Frazier 1801 North Beauregard Street Alexandria, Virginia 22311	1
7.	Commander ATTN: CDR Cable, Code 5115B Naval Air Systems Command Washington, District of Columbia 20361-5110	2
8.	Mr. Thomas Kodey, Code MD 0532 IBM Corporation Owego, New York 13827	1
9.	LT Glenn C. Doyle, USN 22 Highland Avenue Rockville, Connecticut 06066	2